LordCore: Energy-Efficient OpenCL-Programmable Software-Defined Radio Coprocessor

Heikki Kultala, Timo Viitanen, Heikki Berg, Pekka Jääskeläinen, Joonas Multanen, Mikko Kokkonen, Kalle Raiskila, Tommi Zetterman, and Jarmo Takala, *Senior Member, IEEE*

***Abstract*— This paper proposes a single instruction multiple data (SIMD) processor, which is programmed with high-level OpenCL language. The low-power processor is customized for executing multiple-input-multiple-output (MIMO) detection algorithms at a high performance while consuming very little power making it suitable for software-defined radio (SDR) applications. The novel combination of SIMD operations on a transport programmed multicore datapath allows saving power on both the execution front end and the back end, leading to very good energy efficiency with a compiler programmable design. We demonstrate the feasibility of the architecture with the layered orthogonal lattice detector and minimum mean-square-error MIMO algorithms, which can be used as a software-defined radio implementation of the 3GPP local thermal equilibrium r11 standard. Compared to other state-of-the-art SDR architectures, the proposed design adds features that improve programmer productivity with an insignificant power and area impact.**

***Index Terms*— Application-specific instruction set processor (ASIP), digital signal processor, multiple-input-multiple-output (MIMO) detector, vector processor.**

I. Introduction

MODERN wireless telecommunications terminals need to adapt to different standards such as 3GPP long term evolution (LTE) [1], wireless local area network [2], Digital Video Broadcast-Terrestrial/Handheld [3], [4], frequency bands, and operating conditions. Hence, the idea of software-defined radio (SDR) is gaining popularity, i.e., radio signals are processed on programmable platforms instead of traditional fixed-function hardware pipelines. SDR processors are challenging to design since multiple-input multiple-output (MIMO) orthogonal frequency division multiplexing (OFDM) links require tremendous processing power compared to earlier single-channel systems. Meanwhile, the processor architecture must fit in the available power budget, which is especially restrictive in passively cooled and battery-powered mobile terminals.

Manuscript received August 18, 2018; revised December 17, 2018; accepted January 30, 2019. Date of publication February 26, 2019; date of current version April 24, 2019. This work was supported in part by ARTEMIS JU (ALMARVI) under Grant 621439, in part by ECSEL JU (FitOptiVis) under Grant 783162, in part by Finnish Funding Agency for Technology and Innovation (StreamPro) under Grant 40142/14, and in part by the Academy of Finland under Grant 297548. *(Corresponding author: Heikki Kultala.)*

H. Kultala, P. Jääskeläinen, J. Multanen, and J. Takala are with the Faculty of Information Technology and Communication Science, Tampere University, FIN-33014 Tampere, Finland (e-mail: heikki.kultala@tuni.fi; jarmo.takala@tuni.fi).

T. Viitanen is with Nvidia, 00180 Helsinki, Finland.

H. Berg is with EPEC, 60100 Seinäjoki, Finland.

M. Kokkonen and T. Zetterman are with Nokia Corporation, 02610 Espoo, Finland.

K. Raiskila is with Senseg, 02600 Espoo, Finland.

Color versions of one or more of the figures in this paper are available online at http://ieeexplore.ieee.org.

Digital Object Identifier 10.1109/TVLSI.2019.2897508

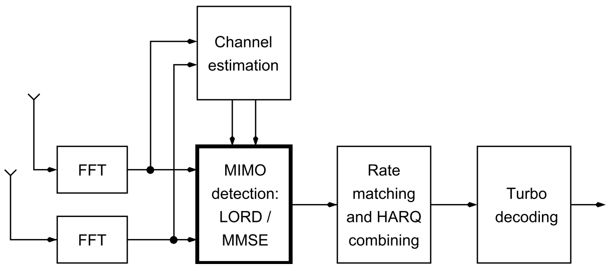

Fig. 1. Main operating blocks of an LTE receiver. This paper concentrates on the MIMO detection part.

In SDR systems, the computational performance is often obtained with increased parallelism in the form of single instruction multiple data (SIMD) computations. Unfortunately, wide SIMD data paths for low-power devices are often designed for maximizing the energy efficiency, which means that they possess nonorthogonal instruction sets and are poor targets for compilers. Such processors require careful manual programming on a low abstraction level, which has resulted in software engineering costs to eclipse the costs of hardware design [5] in system-on-chip projects.

Fig. 1 shows the main operating block of an LTE receiver. One of the most computationally intensive parts of modern wireless communication systems is MIMO detection in the MIMO-OFDM receiver. Of the various proposed detection algorithms, the maximum likelihood (ML) detection provides the best bit error rate (BER) performance, but is impractical to implement as it is based on an exponential-complexity exhaustive search of all the possible digitally modulated symbols. In practice, ML detection is either approximated by algorithms, which limit the symbol search space, or is completely replaced with linear detection, thus trading off equalizer performance to complexity. In general, various MIMO detection algorithms form a tradeoff curve between BER and computational complexity. The minimum mean-square-error (MMSE) [6] equalizer is a representative of linear equalizers, while layered orthogonal lattice detector (LORD) [7] is a suboptimal ML equalizer with deterministic complexity (latency) and soft-output generation complexity, which is linear to the number of transmission antennas.

In an SDR system, the error rate/computation complexity tradeoff gives an interesting opportunity for runtime link adaptation. Close to a base station, where the signal-to-noise ratio (SNR) is high, a simple detection algorithm is sufficient to obtain the required data throughput. At the cell service boundary with a lower user allocation and modulation order, the link may be adapted to use a more complex algorithm. Therefore, the terminal benefits from programmable computing resources such that it can easily run a simple detection at full data rate or a complex detection algorithm at a reduced rate.

In this paper, we propose a novel SDR processor architecture that can be programmed using a high-level programming language (OpenCL [8]) and supports wide-SIMD half-precision floating-point arithmetics. The architecture provides world class power performance while reducing the programming complexity via its use of a high-level parallel programming language and avoiding the need for manual result scaling as with fixed-point arithmetics.

The proposed processor utilizes a transport triggering programming model [9], thus the cores can be classified as transport triggered architecture (TTA) processors. The transport triggering paradigm provides significant energy savings especially on register file accesses. The processor achieves the combination of high performance and programmability using a high-level language (OpenCL C) by means of the following novel architectural features as follows.

1) TTA datapath with wide SIMD operations, which allows processing of several operations with a reduced number of instruction bits resulting in enhanced energy efficiency.
2) Homogeneous multicore TTA design supporting task-level parallelism, which enables improving the throughput and hiding longer memory latencies.
3) Half-precision floating-point arithmetic for efficient implementation of SDR algorithms, which provides good numerical performance without needing the manual signal scaling.
4) General operation set with only a few special operations making the architecture an easier compiler target to enable extended flexibility and applicability.

The proposed architecture is evaluated by means of instruction-level simulations as well as application-specific integrated circuit (ASIC) synthesis and layout. The flexibility of the proposed processor architecture is shown by implementing two different MIMO detection algorithms with link adaptation in mind: MMSE and LORD.

The rest of this paper is organized as follows. Related work is discussed in Section II. Section III introduces the MIMO algorithms used as the primary applications guiding the processor design, and details their OpenCL implementation. Section IV describes the hardware architecture in the proposed processor. Section V presents the evaluation of the architecture and compares it to previous MIMO accelerators. Section VI concludes this paper.

II. Related Work

Traditionally, MIMO decoding is realized as a custom fixed-function hardware unit. For example, an MMSE detector for 4×4 MIMO and 64-QAM on 90-nm ASIC technology is reported in [10]. The implementation achieves 757 Mbps with total power consumption of 189 mW on a chip area of 1.5 mm^2. This translates to power efficiency of 250 pJ/bit. Another hardware-based MIMO detector is described in [11]. On 65-nm process, it achieves 1044 Mbps with total power consumption of 59 mW, meaning energy efficiency of 11 pJ/bit. Two different hardware implementations of least candidate search (LCS)-LORD detector are presented in [12]. A high-performance fixed-function hardware for MIMO detection is presented in [13]. However, these designs do not support postmanufacture functionality updates, which is an essential feature in the proposed solution.

A LORD detector on nVidia Geforce FX 1700 graphics processing unit (GPU) is reported in [14]. MIMO detectors have been reported on Nvidia Tesla C1060 GPU in [15] and Nvidia Geforce 560 Ti GPU in [16]. These GPU chips have high power consumption implying that they would not be suitable for mobile handset implementations.

Many SDR processors use SIMD architecture to achieve high throughput while reducing power consumption compared to generic GPU-based solutions. However, the energy consumption is still much higher than in fixed-function hardware-based solutions. One of the first SIMD architectures for SDR is signal-processing on-demand architecture (SODA) [17], which has 32 lanes of 16 bits for the total SIMD datapath width of 512 bits. The power consumption due to the vector register file is a recurring concern in SIMD radio processors. To reduce the number of required vector register file ports and to save power on accesses to vector register file, SODA opts to allow only one vector instruction per cycle. A long instruction word (LIW) execution with multiple parallel operations would require more register file ports, while the vector register file was already the largest single power consumer.

An improved revision of SODA, Ardbeg [18], attempts to improve vector arithmetic logic unit (ALU) utilization by permitting LIW operation, but provides only three register file read ports and two write ports, and limits the available instruction combinations accordingly. For example, it is possible to schedule an add and a load instruction on the same cycle, but not two simultaneous adds, which would require four read ports. The VEGAS architecture [19] discards the register file entirely and loads vector operands from scratchpad RAM, but in the context of a softcore field-programmable gate array (FPGA) implementation. The AnySP [20] architecture as well as Waeijen *et al.* [21] introduce a forwarding network with explicit bypassing in order to save power in vector register file. This approach takes advantage of the fact that many register values are only consumed once after being written to the register file. For such values, it is possible to bypass the value from the producing function unit to the consuming unit through the forwarding network and eliminate a write to the register file. Typically, ca. 50% of register file writes can be eliminated in this way, saving substantial power.

There are other techniques for optimizing the RF power consumption and complexity in very LIW (VLIW) style signal processors proposed in the past: PAC [22] utilizes a ping-pong register file, which consists of two clusters, each having their own private RF, and a common dual-banked register file where one bank is always connected to one cluster. This allows each RF bank to do with a small number of ports. Zalamea *et al.* [23] propose a hierarchical RF structure. This reduces traffic to the large central RF when most of the accesses hit the smaller and more energy efficient inner level RF.

In [24], a custom processor tailored for MIMO algorithms is proposed, which shows good energy efficiency for a programmable solution. However, the processor has a limited instruction set and is, therefore, programmed in assembly language making algorithm implementation, optimization, and debugging burdensome. Also, this processor does not perform the channel preprocessing part of the MIMO detection.

The napCore is a floating-point SIMD signal processor for SDR baseband processing [25]. The solution is area efficient but it still uses highly customized data types, which means that widely used high-level programming languages such as C cannot be efficiently used to program it. Tomahawk [26] is a multiprocessor SoC containing a baseband processing module (BBPM), which has programmable signal processors for MMSE MIMO equalization, symbol detection, and turbo decoding.

A coarse-grained reconfigurable array (CGRA)-based application-specific instruction set processor (ASIP) for MIMO detection is proposed in [27]. CGRA allows high performance and efficiency with programmable control, but CGRAs are challenging to program as they require software pipelineable loops for efficient utilization. Furthermore, their proposed CGRA is not capable of executing generic programs written in high-level languages efficiently as it computes on highly nonstandard 2×18b and 2×23b fixed-point complex numbers, which cannot be described by popular high-level languages such as C or OpenCL C. The log-likelihood ratio (LLR) calculation is handled by a fixed hardware block, not by a programmable processor, which may make the ASIP unsuitable for some other detection algorithms.

ADRES is a processor template for baseband processing [28]. It consists of a VLIW processor and a coarse-grained reconfigurable matrix. In [29], an MIMO detector for ADRES-based processor is proposed, which has a 256b SIMD datapath consisting of eight lanes, where each lane has complex-valued arithmetic units. However, their multi-tree selective spanning-based MIMO detection supports only hard-bit decision output, not soft-bit output. Another related ADRES-based processor for radio baseband processing is presented in [30].

A custom multicore TTA processor for lattice reduction is proposed in [31]. The system is based on a pipelined multicore, where each core executes each stage of their multistage algorithm. The system, however, does not perform the whole MIMO signal detection, only the part of lattice reduction, which can be used in MIMO detection implementations.

Most of the previously discussed programmable solutions require extensive and laborious programming work to reach high performance as the processor architectures are not designed for high-level language compilers. Unlike the previously discussed solutions, the proposed architecture allows programming with higher level OpenCL programming language, which is much easier for programmers, and it allows the code to be ported from different platforms with minimal modifications. The proposed architecture can run many other workloads such as other parts of the SDR radio stack and even applications from other application domains. In [32], we reported a preliminary version of the design where it was used for an image processing application. This showcased the cross-domain programmability of the proposed approach. The proposed architecture has in the order of 1000× better energy efficiency than traditional GPU-based solutions. However, the peak performance is much lower than on the advanced GPUs. Similar to [17] and [18], the proposed architecture can also operate with fewer register file ports than the peak usage of the execution units would require, but it exploits specific optimizations to alleviate its impact to the execution performance; TTA's software bypassing and operand sharing allow using RFs with fewer ports [33], reducing the power consumption even further. RF ports are especially costly in wide SIMD RFs. The result is a design with fewer register file write ports than in Ardberg [18], while still allowing more instruction-level parallelism for issuing multiple different (vector or scalar) operations in concurrently.

III. OpenCL Implementation of the MIMO Detectors

The proposed processor architecture was produced as a hardware–software codesign with two different MIMO detection algorithms as the primary benchmarks guiding the process; MMSE [6] and LORD [7]. These algorithms were implemented and optimized as OpenCL applications.

A. MMSE and LORD Algorithms

MMSE is a simple and fast high bitrate algorithm with lower accuracy. It is typically used when the device is close to the base station and the SNR is high. MMSE equalizer is a widely employed linear equalizer in wireless communications. The MMSE detector consists of two parts for each layer: the equalization, which includes some heavy matrix computations including matrix inversion of the channel matrix, and log-likelihood soft output generation, which consists of multiplications, subtractions, and minimal value selections. The amount of these operations in the equalization part scales roughly by $O(N^3)$ to the amount of layers. The amount of these operations in the soft output generation scales by $O(N)$ to the number of layers and by $O(2^N)$ to the number of bits per symbol.

LORD is a more complex and calculation intensive algorithm than MMSE. It is an approximate method for joint Max-Log-maximum *a posteriori*-based detection which directly produces the soft-bit values and is typically used for lower bitrate transmissions when the SNR is lower. The algorithm contains two main steps: first QR decomposition of the channel matrix is calculated and the signal is preprocessed by multiplying it by the conjugate transpose of the Q matrix from the QR decomposition. Then the soft-output bits are generated by searching the minimal values of differences between the current layer and combined interference of other layers. This is performed by series of complex multiplications, subtractions, and min operations. The amount of these calculations scale by $O(2^{(2N)})$ to the number of bits per symbol and they are performed for every layer.

Both of the algorithms have to perform the detection of the signals by comparing the signal to so-called constellation points in the complex plane, and finding which the signal corresponds to. QPSK modulation carries two bits per carrier, thus it contains 2×2 of those constellation points. QAM-16 contains 4 bits/carrier, and contains 4×4 grid of constellation points. QAM-64 contains 6 bits per/carrier for 8×8 grid of constellation points, and QAM-256 contains 8 bits per carrier for 16×16 grid of constellation points.

B. OpenCL Implementation

As the processor was produced with a processor hardware–software codesign methodology, a portable software implementation was developed to be compiled to various design points of the processor created during the codesign process. For this design, LORD and MMSE were implemented using the OpenCL C language, which allows using vector data types to execute the same code on many SIMD lanes of the processor and supports easy parallelization of workloads over multiple cores.

Traditionally fixed-point arithmetic has been used for low-power signal processing, which implies that a careful signal level scaling is needed when implementing the algorithms. This does not suit well on programming with high-level languages like OpenCL. Recent studies suggest that low-precision floating-point arithmetic is a competitive approach in communications applications in terms of energy efficiency [34]. Therefore, we exploited the half-precision floating-point arithmetic [35], which is now included in the IEEE-754-2008 standard [36]. The half-precision floating-point representation contains a 10b mantissa, 5b exponent, and a sign bit resulting in a 16b data type. It is notable that this number representation provides the same dynamic range as a 33b fixed-point representation. As two half floats can be stored in the space of a single-precision float, the number of SIMD lanes can be doubled with the same total bit width without any extra cost. This allows doubling the throughput of the processor core without increasing the datapath width, which saves the power consumption.

In order to reach the required computing performance, we defined that each subcarrier is executed in an SIMD vector lane of its own and, therefore, the algorithm is executed for 32 parallel subcarriers per core. We did not modify the original MMSE and LORD algorithms to extract more data-level parallelism from inside a single data stream, but instead process 32 separate data streams in parallel. This can be done, thanks to huge number of subcarriers in all wireless communication modes. The group of 32 subcarriers that execute in one core at a time form one single-work-item OpenCL work group. Several such work groups can execute concurrently on the multiple cores of the processor, thus enables utilizing task parallelism from the software side. With two layers, the number of subcarriers calculated concurrently is 4096. With four layers on MMSE, the number of subcarriers calculated concurrently is reduced to 1024 to save memory.

The constants containing the constellation point coordinates are kept in vector registers and indexed from there with specialized shuffle instructions. These instructions are not MIMO detection specific and they are also useful in other workloads. The special shuffle instructions are called with special intrinsics that are exposed to the programmer as OpenCL C extensions. Keeping the constellation constants in the vector registers allows precomputing them while still omitting loads from memory to use the values. In order to find the sweet spot, we varied the SIMD width. This requires only relatively small manual changes to the program; only the SIMD data types were changed and the shuffle intrinsic calls were modified. The softbit loops of the algorithms were heavily unrolled to reduce the number of load and store operations needed in the inner loops and to increase the instruction-level parallelism.

For the matrix inversion in 2×2 mode, the MMSE implementation uses Cramer's rule [37] to minimize the number of operations needed for the calculations. In 4×4 mode, Cholesky decomposition is used, which is partially manually unrolled to achieve better performance. The LORD code uses QR decomposition for preprocessing. The algorithms use reciprocal and reciprocal square root operations. Lower accuracy approximations for these were obtained by using the Newton–Raphson method [38] with an initial value. This provided good performance without the need to add large power-hungry function units for vector division and square root.

IV. LordCore Architecture

In order to support the OpenCL implementation discussed in Section III-B, we defined the architecture of the SDR processor core. The proposed processor core, referred to as LordCore, is based on the power-efficient transport triggering paradigm, which we describe briefly in the next.

A. Transport Triggered Architecture

In transport triggering paradigm [39], a program defines only data transports between various resources of a processor and the operation execution occurs as a side effect for a data transport. In this sense, transport triggering reminds the traditional dataflow model of execution. In a TTA processor, the datapath buses are exposed to the programmer; the programmer schedules data transfer along the buses. Operands to a function unit are moved via an interconnection network to input ports of a function unit and one of the ports is dedicated as a trigger. Whenever data are moved to the trigger port, the operation is triggered, i.e., executed. Therefore, the program defines only the data move on the interconnection network, thus the TTA processor has only one instruction: move. In Fig. 2, an example of a TTA processor is illustrated. The processor contains an interconnection network with five transport buses implying that at most five data transports can be executed in parallel, i.e., each instruction contains five move slots. The figure illustrates execution of an instruction with three parallel moves, i.e., instruction has three move slots and two of the buses are not used during the clock cycle

$$\#4 \rightarrow \text{ALU1.i0.ADD}; \quad \text{RF2.r3} \rightarrow \text{ALU1.i1}$$
$$\text{RF0.r1} \rightarrow \text{LSU0.i0.STW}.$$

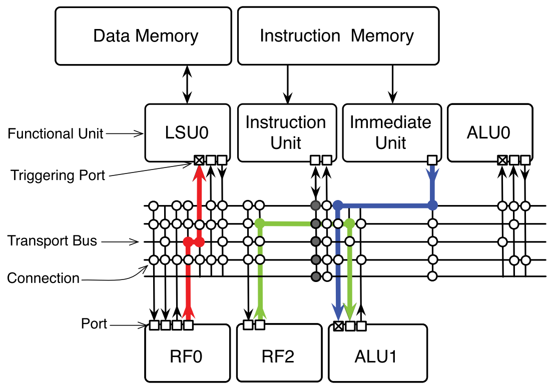

Fig. 2. TTA processor organization. Different colored arrows show three different data transfers being performed in the processor datapath simultaneously.

On the first transport bus, an immediate value is moved to input port 0 of the function unit ALU1. The immediate value is actually obtained from the immediate unit, which has only one output port. The move to trigger port carries information about the operation to be executed; opcode ADD is transported to function unit along with the operand. This data transfer is shown in blue color in Fig. 2. The second bus transports an operand from register r3 through the output port 0 of the register file RF2 to the input port 1 of the ALU1, which is shown in green color in Fig. 2, and the third bus transports an operand from r1 in the register file RF0 to the input port 0 of the load–store unit (LSU) LSU0, which is shown in red color in Fig. 2, The third move contains an opcode indicating that the transported word is to be stored to memory. The actual store address has been defined earlier by another move to port 1 of the LSU0. The remaining two move slots are empty, thus the corresponding two buses, thus they can be considered executing a no-operation code.

Compared to traditional "operation programmed" VLIW architectures, where the instruction set specifies operations and data transfers occur as side effect of instruction execution, the TTA programming model has the benefit that the register file bypasses are explicitly programmed ("software bypassing") [40], and all the operands of operations do not have to be read in the same clock cycle. Similarly, the computed results do not have to be read to the destination register file on the same cycle they are produced, and the result write to a register can be totally omitted of the result is bypassed directly to some another operation. Also, in case same value is used multiple times by same function unit, the value does not have to reread from the register file every time. This is called operand sharing [41]. These optimizations allow the use of smaller register files with fewer read and write ports [33] in multiissue designs.

In TTA processors, the register files and function units are fully decoupled from the rest of the architecture due to the customizable interconnection network, it is easy to design new processors by varying the processor resources. The transport triggering paradigm works, especially well for wide SIMD datapaths, thanks to SIMD instructions saving instruction bits per operation and thus instruction fetch power, which is typically the pitfall of TTA processors, while the reduced datapaths in TTA-type processors save power on the execution back end, where most of the power of SIMD processors are spent.

The buses of a TTA processors may be guarded by predicates, which means that each of the moves in a TTA instruction may be conditional. In case the predicate evaluates to false, the move is converted to a no-op when it is decoded. This allows utilizing the if-conversion compiler optimization to remove branches from the code, and, e.g., an optimization where conditional branch delay slot instructions are filled with moves from the branch target location. The drawback is that it may increase the instruction width as there may be additional predicate fields per instruction needed, and extra control signals are needed to implement the predication. However, these predicates are optional design choices in a TTA-based coprocessor.

B. Processor Design and Code Generation Workflow

The processor codesign process was carried out with TTA-based codesign environment (TCE) tools [42]. The design flow is described in more detail in [43] and [42]. For providing the OpenCL support to the designed processor, we utilized our earlier work, Portable Computing Language (pocl) [44], a general-purpose OpenCL implementation exploiting the low-level virtual machine (LLVM) compiler framework [45]. The OpenCL standard supports only 16-wide vector data types, but we defined a new vendor extension to support up to 32-wide vector for increased throughput. Adding support for 32-lane wide vectors required only a few lines of code in pocl as the LLVM compiler framework used by pocl already supports a wide variety of different sizes of vectors.

The compiler consists of two main parts: 1) the pocl OpenCL implementation that handles the kernel parallelization and provides an OpenCL C kernel built-in library implementation and 2) the retargetable TTA compiler provided by the TCE toolset which has a compiler that performs the final instruction scheduling for each TTA variation. Both parts are based on LLVM, and the data transfer between pocl and TCE compilers is done in the LLVM bitcode format. This split is, however, transparent to the programmer: the programmer just uses OpenCL and pocl automatically calls the TCE compiler in the background when the chosen device is a TCE coprocessor. Li *et al.* [46] also reported a similar compilation flow, but instead of a retargetable TTA target they had a VLIW processor.

As OpenCL is a heterogeneous programming standard, it uses an intermediate higher level program exchange format for portability. In this case, we used the textual OpenCL C. The use of intermediate format is useful in reducing the binary compatibility burden inherent of statically scheduled machines such as VLIWs or TTAs: openCL programs are typically compiled "online" at launch time by the host runtime and cached per device.

For the design proposed here, the TCE toolset was extended to support SIMD operations and instruction scheduling algorithm from [47] was added. Fig. 3 shows the typical workflow of the TCE toolset for OpenCL programs. Architecture definition file describes a TTA processor

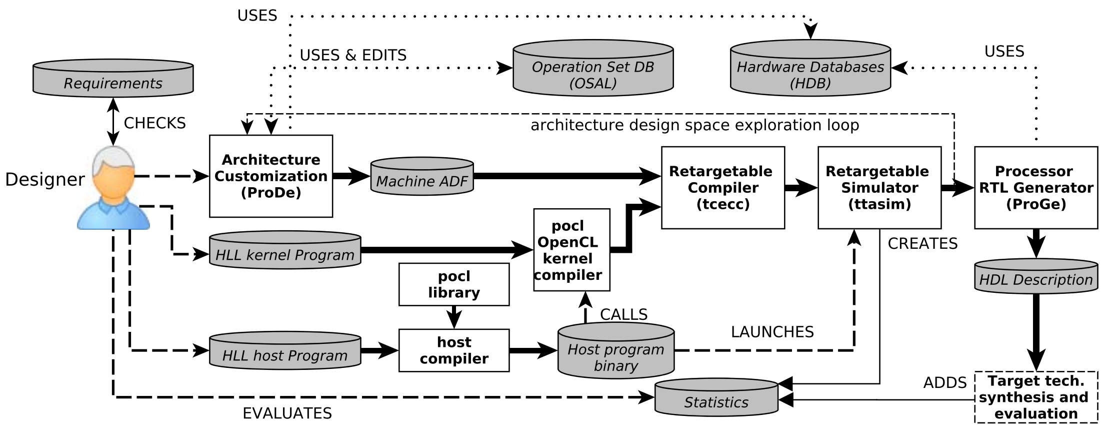

Fig. 3. TCE design flow for OpenCL input programs. The programmer writes both kernel and host code, and creates an ADF of the TTA architecture. The host code is compiled and linked with the pocl library. When the host program is being run, it compiles the OpenCL kernel using the retargetable compiler of TCE, and simulates the program with the TCE's simulator. The results of the simulation can be used for iteratively optimizing the code and the architecture.

architecture in the level of programmer visible details as it contains information about the functional units, register files, and internal buses and their connections in each design. The TCE compiler (tcecc) inputs this architecture description file (ADF) as a parameter and adapts to the target on the fly. The processor generator tool of TCE uses this information to automatically generate the hardware description language that can be used to implement the processor, either using predesigned function unit/RF components or automated function unit generation. The ADF can be easily edited by a graphical design tool (ProDe) which allows inspecting a visual representation of the architecture at hand.

The concrete design process for the proposed system went roughly as follows. First clean OpenCL implementations with 32b float calculation precision of the algorithms were created. Then, a simple and slow 32b float-supporting scalar TTA architecture was generated with the TCE toolset. Then a 8 × 32b float SIMD TTA architecture and vectorized version of the code were created. This architecture was also used to analyze bottlenecks of the system and optimize the implementation. After some optimizations, a 16 × 16b half-precision SIMD TTA architecture was created and the code was ported to use half-float vectors instead of float vectors. Simulations were used to verify the 16b version. The simulations also showed promise of considerably better achievable performance with even wider SIMD, so the SIMD width was extended to 32 lanes and support for 32-lane vectors was enabled in pocl. After the final SIMD configuration was in place, some optimizations were made to both the architecture and the code, before proceeding to the actual hardware design. A slightly modified version of the algorithm described in [48] was used to optimize the interconnect network of the processor and achieve a 128b instruction word. Finally, the hardware implementation of the architecture was produced and synthesized for both an FPGA prototype and the targeted ASIC technology.

C. Organization of Single-Core LordCore

The LordCore architecture has vector and scalar datapaths as shown in Fig. 4. The vector datapath is a 512b wide SIMD datapath for high-performance computations. The SIMD datapath can perform computations with 32 half-precision floating-point values in parallel. The scalar part is a 32b integer datapath for address and control calculations. The design is multicore-ready including a hardware mutex unit for efficient intercore synchronization. The details of the core are discussed in the following.

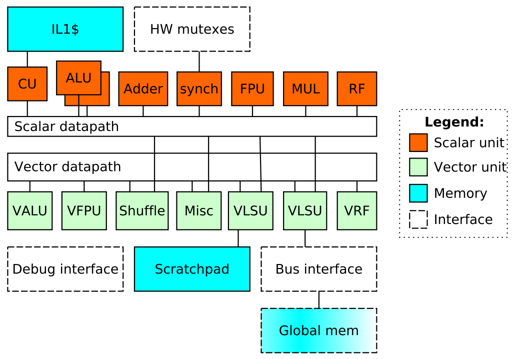

Fig. 4. Single-core LordCore architecture with system-level interfaces. The TTA datapath is illustrated in higher detail in Figs. 5 and 6. The multicore system architecture is shown in Fig. 7.

1) Execution Units: The scalar datapath contains two scalar ALUs with multiple basic operations and a simple address generation adder, a multiplier, a floating-point conversion unit, synchronization unit, and a timer unit. There are also I/O units for debugging purposes and address and scalar data ports to the LSUs. The SIMD datapath consists of a 32 × 16b SIMD floating-point multiply accumulate unit, which can also execute separate add or multiply operations, a 32 × 16b integer ALU, and vector floating-point comparator and conversion units. There are also two execution units for transferring data between the SIMD lanes and between the scalar and SIMD datapaths. Both the vector comparison unit and the vector ALUs contain min and max operations for half-float data types because these operations are executed concurrently with both vector floating-point operations and vector integer operations in the algorithms. The half-precision floating-point units are considerably smaller, faster, and more power efficient than the traditional IEEE-754 single-precision floating-point units.

TABLE I
INSTRUCTION LATENCIES AND THROUGHPUTS OF DIFFERENT INSTRUCTION CLASSES. THROUGHPUT MEANS THE MAXIMUM AMOUNT OF INSTRUCTIONS EXECUTED PER CLOCK CYCLE

Instruction classes	Latency	Throughput
add,sub	1	3
other scalar ALU	1	2
multiply	2	1
call/jump	4	1/4
Load from local memory	3	1
Load from global memory	10	1/3
fp16 vector fma/add/mul	3	1 (x32)
fp16 vector comparison	1	1 (x32)
fp16 vector min/max	1	2 (x32)
fp16 $\longleftrightarrow$ i16 vector conv	2	1 (x32)
i16 vector ALU	1	1 (x32)
shuffle	1	1
vector broadcast, build	2	1

The vector datapath contains shuffle operations, which can rearrange data from 8- or 16-wide vectors to 32-wide vectors. This limited shuffle is used for table lookups from 8- or 16-element size constellation tables, which are loaded into vector registers. There is also an operation to copy a single vector lane into all the vector lanes of the result. This operation could have also been done with a generic shuffle but copy simplifies programming as no shuffle mask is needed. Table I summarizes the datapath operation resources.

Special complex-valued arithmetic units were considered but they were omitted as such operations would have allowed transferring each complex-valued operand to the execution unit only once and performing two multiplications with each transferred part of complex-valued operand with only one register read. It was found that the operand sharing optimization of the processor already removed the second register access in most of the cases, thus the benefit from complex-valued arithmetic would have been minor and the fused multiply-add (FMA) operations were very well suited for the complex-valued computations. Special complex-valued arithmetic units would also have required writing special intrinsics to use them and they would not have been as flexible as the scalar FMA units.

2) Register Files and Internal Buses: For scalar operands, there is a general-purpose register file with 32×32 b registers and three read and three write ports. There is also a boolean register file with two 1b registers and a read and write port. The boolean register file can be used as a predicate to all the data transfers inside the core, allowing both conditional moves and other predicated operations. However, as these predicates are only 1b scalars, lane mask-based predication is not possible. The scalar side contains three internal buses as illustrated in Fig. 5. There is also a vector register file with 32 registers of 512 bits with three read ports and one write port. The vector side contains four internal buses as shown in Fig. 6.

3) Scalability: The processor core was designed to enable throughput improvements via task-level parallelism. The memory architecture allows scaling to multiple cores and the SIMD width of the processor can easily be scaled. We selected 32 lanes as the default SIMD width since such a relatively wide SIMD allows more work to be done per instruction bit. Wide SIMD also reduces the relative instruction fetch energy, while still keeping the individual cores reasonable small for physical implementation. A smaller single core with 8 or 16 SIMD lanes could be used for lower throughput systems such as LTE-M, while multicore 64-lane versions could be used to extend the throughput for future communication standards or even other application domains.

D. System Architecture

The proposed processor core is integrated into a multicore accelerator as shown in Fig. 7 for a four-core configuration. As seen in the figures, the processor cores are connected with an Advanced eXtensible Interface (AXI) interconnect. OpenCL defines several logical memory spaces, including global, local, and private memories. In the style of private local memory processors such as IBM Cell, we included a tightly coupled scratchpad memory with each core. The OpenCL local and private memory spaces and the heap are mapped to this scratchpad. The global memory is shared between the cores in the system and it is also used to store input and output data. This memory is sized at 448 KB, which is the amount required to contain all the input and output buffers, global data buffers and code for the implemented algorithms for calculating 4096 subcarriers in parallel in LORD or 2×2 MMSE.

After the test workloads were optimized to use the local scratchpad, there is practically no data access locality, i.e., every data element in global memory is accessed once. Consequently, there is no benefit for including a data cache, thus the bus interface is connected directly to the LSU. However, special design is required in the LSU as the bus interface has a long and varying access latency compared to the scratchpad. In order to achieve a reasonable data throughput, an LSU has to handle multiple in-flight requests. This is done by setting a fixed architectural latency and storing prematurely arriving data in a ring buffer. If a store takes longer than the architectural latency, the core stalls. When the ring buffer size is equal to the architectural latency, it is possible to issue a memory operation at every cycle, however, the ring buffer then becomes comparable in size to the vector register file. We found through experiments that a four-entry ring buffer and 10-cycle latency are a good tradeoff between the hardware complexity and performance.

LordCore fetches instructions from the global memory. For improved access times, each core has a directly mapped instruction cache with a capacity of 2048 instructions (32KB), which is sufficient for the MIMO detection kernels. As a result, there are no cache misses except for a warmup period during the first time a given kernel is executed. There is no hardware-based coherence in the instruction caches as they are read-only from the core point of view. An external invalidate signal is used for reprogramming. This instruction memory structure was selected to allow high performance for the MIMO kernels while still allowing running workloads with greater instruction memory requirements without wasting transistors for fast and large instruction memories. Each processor core has an attached hardware debugger which doubles as a control interface. In the course of normal use, the debugger is used to

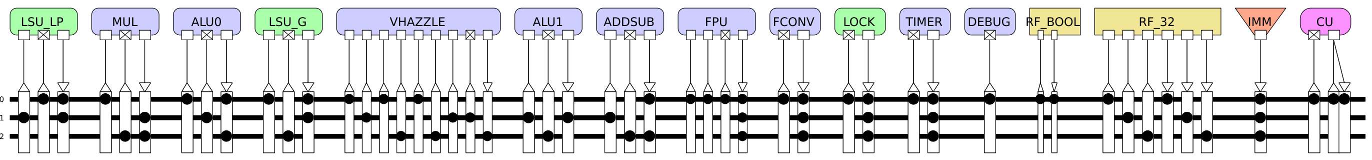

Fig. 5. Scalar datapath of the LordCore processor.

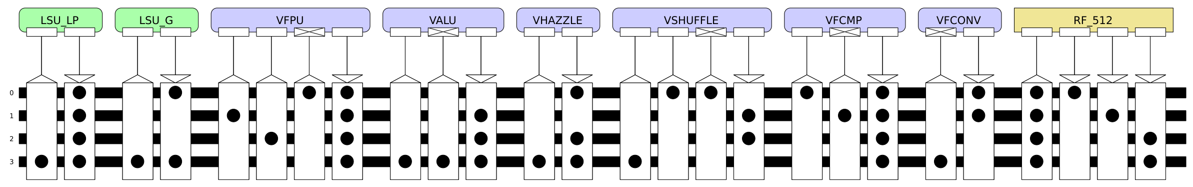

Fig. 6. Vector datapath of the LordCore processor.

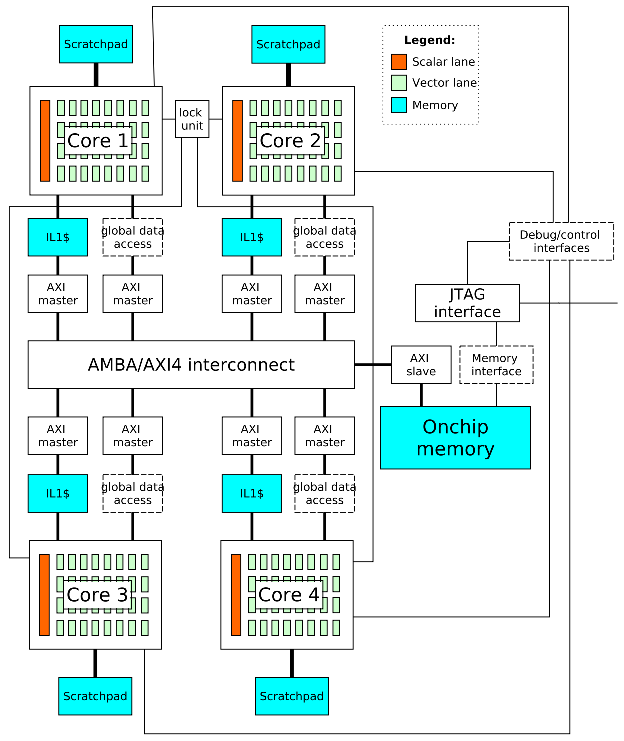

Fig. 7. Organization of a quad-core LordCore system.

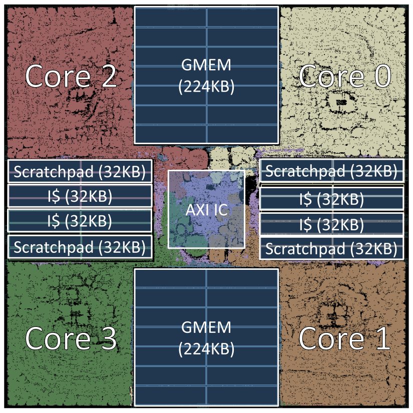

Fig. 8. Quad-core ASIC layout.

invalidate caches, start and stop execution, and set instruction and data memory windows. The memory windows specify the locations of instruction and data memory as ranges in the larger AXI address space, and function as simple memory protection. Moreover, the debugger can be used to set breakpoints, perform stepping execution, and examine the architectural state of the processor, including the values on each transport bus. Finally, a hardware lock unit [49] is connected to each core to synchronize primitives without expensive global memory traffic. The unit is mainly used for synchronizing the work distribution between cores: each core pulls OpenCL work items from a work item queue guarded with a mutex, which is implemented using the lock unit.

V. Evaluation

In order to verify correct functionality, we carried out RTL simulations of single-, dual-, and quad-core configurations running the MMSE and LORD workloads, and compared outputs against a cycle accurate simulator. Moreover, the single- and dual-core systems were prototyped on a Xilinx Zynq XC7Z045 FPGA at 39-MHz clock frequency, with a reduced global memory size to fit on the FPGA. For evaluation purposes, the quad-core system was synthesized, placed, and routed with Synopsys tools using a commercial 28-nm fully depleted silicon-on-insulator process technology. Clock gating and multithreshold voltage optimizations were enabled in the synthesis. Operating conditions were set to 1.0-V supply voltage and 25 °C temperature. In order to cope with I/O pad limitations, a JTAG interface is used to fill the global memory and access the debugger. The routed design achieves a clock rate of 968 MHz and has a cell area of 2.47 mm^2 at an utilization of 71%. The final layout is shown in Fig. 8.

The power estimation was performed with Synopsys IC Compiler based on switching activity extracted from RTL simulations. Each kernel execution exhibits a warmup period during which the instruction caches are filled; this period was omitted from the extracted switching activity in order to obtain power figures representative of prolonged execution. Table II shows the estimated power and energy efficiency of the quad-core system. In general, 22.0% of the power is consumed by vector execution units, 8.3% by the vector register file, 2.1% by the scalar ALUs and scalar registers, 8.5% by the interconnect network, 26% by the global memory, 16.9% by

TABLE II
SIMULATED POWER CONSUMPTION OF LORD AND MMSE ALGORITHMS ON THE LORDCORE AT 900-MHz CLOCK RATE

Algorithm	ntx / nrx	Modulation	Power [mW]	E/bit [pJ/bit]
LORD	2 / 2	64-QAM	259	906
LORD	2 / 4	64-QAM	269	1008
MMSE	2 / 2	64-QAM	226	122
MMSE	2 / 2	256-QAM	241	122
MMSE	4 / 4	64-QAM	245	367

TABLE III
SIMULATED THROUGHPUT PERFORMANCE OF LORD AND MMSE ALGORITHMS RUNNING ON THE PROCESSOR ON DIFFERENT MODES AND DIFFERENT CORE COUNTS AT 900-MHz CLOCK FREQUENCY

algorithm	ntx/nrx	modulation	single core [Mbps]	dual-core [Mbps]	quad-core [Mbps]
LORD	2 / 2	QPSK	122	226	445
		16-QAM	125	241	472
		64-QAM	74	146	286
LORD	2 / 4	QPSK	85	161	314
		16-QAM	103	200	391
		64-QAM	69	136	267
MMSE	2 / 2	QPSK	218	380	703
		16-QAM	427	746	1382
		64-QAM	549	976	1850
		256-QAM	571	1043	1979
MMSE	4 / 4	QPSK	60	117	229
		16-QAM	119	232	454
		64-QAM	172	335	639
		256-QAM	210	409	784

instruction caches, and 16.2% by the local scratchpad memory while running the LORD algorithm.

The effect on the TTA-specific optimizations on the power consumption was analyzed in preliminary work in [50]. The TTA-specific optimizations provided on average 18% decrease in power consumption in these workloads.

A. Execution Time

Both the algorithms were analyzed on the processor cores at 900-MHz clock rate. The LORD algorithm was executed with two layers: with two and four receive antennas and with QPSK, 16-QAM, and 64-QAM modulations. The MMSE algorithm was executed with both the 2×2 and 4×4 modes and QPSK, 16-QAM, 64-QAM, and 256-QAM modulations. Table III shows the results of these performance benchmarks. The LTE category four requirements (2×2 MIMO, 64-QAM modulation, 150 Mbit/s) can be achieved with a single core when using the MMSE algorithm or with three cores when using the LORD algorithm, which has better detection performance. The quad-core cluster can exceed the performance requirements of LTE r11 (4×4 MIMO, 64-QAM modulation, 600 Mbit/s) when using the MMSE algorithm. The analysis shows that stalls due to cache miss or bus contention consists an average of 2.2% of the runtime with four cores and 0.7% with two cores.

1) Scaling Between Algorithm Options: In the LORD algorithm, the number of operations in the equalization phase is independent of the modulation, but the number of operations in the softbit selection phase with two transmit antennas is proportional to $O(2^{2b})$, where b is the number of bits in either axis. The performance in the LORD increased when moving from QPSK to 16-QAM modulation as the amount of transferred bits doubled from 2 to 4 bits per carrier, as the equalization phase consumed most of the execution time, and the softbit selection phase, which scales superlinearly consuming only a small fraction of the execution time.

With 64-QAM modulation, the softbit selection loops begin to dominate the execution time due to their superlinear time scaling, leading to a lower throughput performance than with the 16-QAM modulation. There is also a further discontinuity in performance at 64-QAM since the QPSK and 16-QAM modulations can be optimized by fully unrolling of the softbit selection loops while still storing all the temporary values in registers.

In the MMSE algorithm, all the computations in the softbit selection loops are carried out independently for every layer thus the amount of computations is only proportional to $O((2^b)*n)$, where n is the rank of the detector, i.e., the number of transmit antennas. This means that the softbit loop execution time of MMSE runs much faster, especially on many transmit antennas and complex modulations. Therefore, compared to the LORD, MMSE spends a much larger portion of its execution time in the equalization phase and the performance increases with more complex modulations.

When using more receive antennas in the LORD algorithm, the number of operations increases in the equalization phase but it does not affect the number of operations performed in the softbit selection phase. With 64-QAM modulation, increasing receive antennas from 2 to 4 reduces performance only by 11%. With QPSK modulation, where more time is spent on the equalization phase, the performance drops by 31% with four antennas. When increasing both the number of receive antennas and the rank of the detector, there is a huge effect on performance with MMSE, as the most computationally intensive part in the MMSE algorithm is the matrix inversion in the equalization. The matrix inversion has a cubic computational complexity with regards to the number of antennas, while a fast algorithm can be used with two antennas.

B. Bit-Error Rate Performance

The BER performance of the LORD and MMSE algorithms was analyzed with simulations; a reference case with 32b single-precision floating-point arithmetic is compared to the proposed processor core exploiting 16b half-precision floating-point arithmetic. The results are uncoded results before the Forward Error Codec (FEC), with the independent identically distributed Rayleigh fading channel model.

After MIMO detection, the softbits are fed to the FEC, which, in case of 4G, is Turbo Decoder [51], one block at a time. In case, the FEC cannot fix all the errors in the block, it is retransmitted. In case of too many blocks have to be retransmitted, the transmission mode is changed to a slower and more reliable transmission mode, and if no blocks need retransmission, the transmission mode is changed to a faster and less reliable one. In our measurements, the optimal balance between reliability and performance is when about 10% of all the blocks need to be retransmitted, and to reach this BER of about 0.01 is enough. However, lower BER can still give

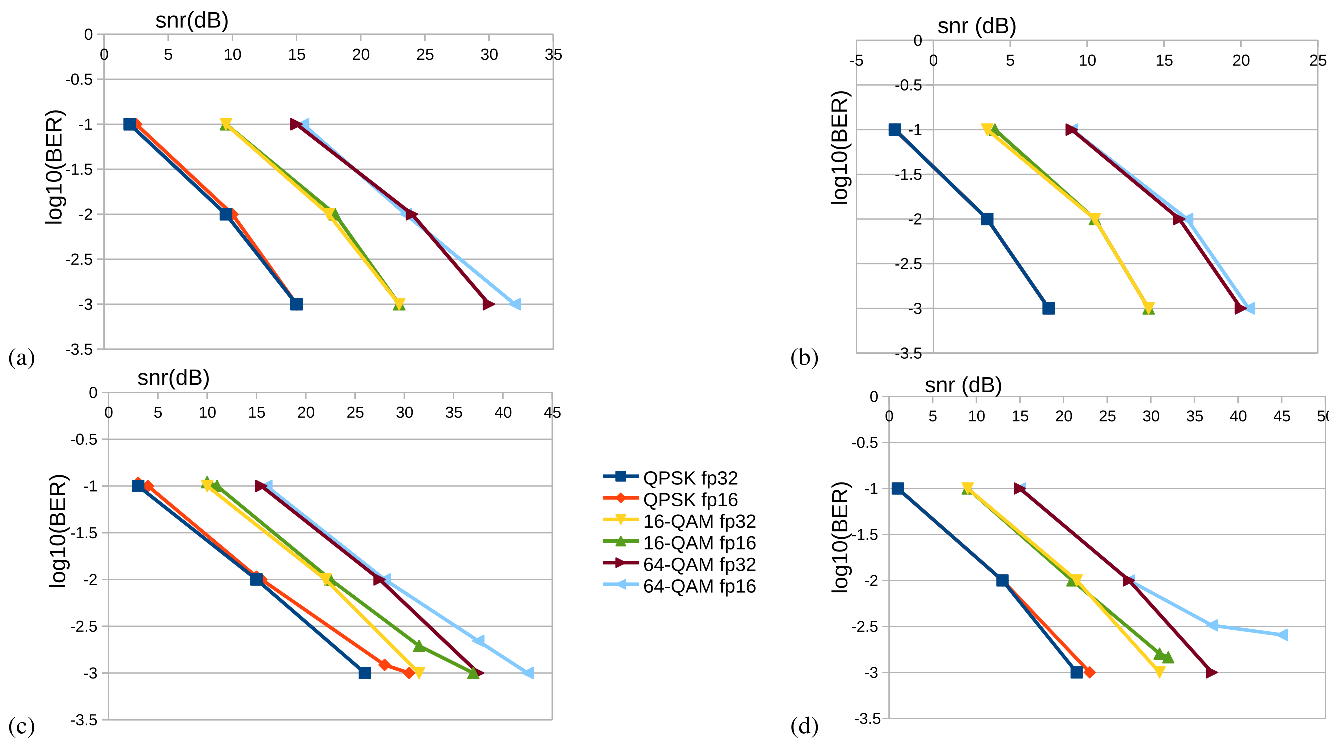

Fig. 9. BER performance of the LORD detector in (a) 2×2 and (b) 4×2 mode and MMSE detector in (c) 2×2 and (d) 4×4 mode. fp32 is the reference code with 32b single-precision floating-point arithmetic. fp16 is the code with half-float arithmetic on the proposed processor.

slight advantages due to less retransmissions if faster and less reliable transmission mode cannot be used.

The BER performance of the LORD algorithm on 2×2 and 4×2 modes is shown in Fig. 9(a) and (b), respectively. In both modes, the performance on the proposed processor with 16b half-precision floating-point arithmetic is typically very close to the performance of 32b single-precision floating-point arithmetic. However, with 64-QAM modulation, slightly better signal quality is required to reach BER value of 0.001.

Fig. 9(c) shows the BER performance of the faster and lower precision MMSE algorithm executed on the proposed processor in 2×2 mode. To reach BERs of 0.1 and 0.01, the required SNR is very close to the reference, but to reach BER of 0.001, considerably better signal quality is required. The BER performance of the MMSE algorithm executed on the proposed processor on 4×4 mode is shown in Fig. 9(d). To reach BERs of 0.1 and 0.01, the required SNR is very close to the reference, but on 16-QAM and 64-QAM modulations, the low precision of the 16b half-precision arithmetic starts to have a considerable effect, and BER never reaches 0.001, no matter how good the signal is.

C. Comparison

The SIC algorithm in [16] is assumed to have roughly the same detection accuracy than the MMSE algorithm, while the nML algorithm is assumed to have similar accuracy as the LORD algorithm. Table IV shows the performance comparison of MIMO detection algorithms, and the characteristics of the platforms executing these algorithms are listed in Table V. The energy consumption figures represent the highest average consumption while running any of the MIMO algorithms, except for the GPUs, which contain thermal design power (TDP) numbers. The actual power consumption of these GPUs while running the codes was not reported, and is likely somewhat less than the TDP, but still in the same magnitude.

Total SIMD/SIMT lanes mean the total amount of data-level parallelism, which the hardware can exploit. Compared to the LORD algorithm running on Geforce FX 1700 GPU [14], even a single LordCore achieves almost 5× performance with over 1000 times better energy efficiency. In the 2×2 16-QAM mode, a dual LordCore is needed for the same performance as the Multipath Trellis Traversal (MTT)-based MIMO detector on Nvidia Tesla C1060 GPU [15], giving hundreds of times better energy efficiency. In case of 2×2 64-QAM with LORD algorithm, a single LordCore outperforms Tesla GPU [15], providing thousands of times better energy efficiency.

A 2×2 MIMO detector in 16-QAM mode with one-way algorithm on Nvidia Geforce 560 Ti GPU in [16] has the same throughput as dual LordCore executing MMSE, but Lordcore has hundreds of times better energy efficiency. In 2×2 64-QAM, a single LordCore executing MMSE reaches 3× throughput compared to Geforce GPU with one-way algorithm, provides thousands of times better energy efficiency. On 2×2 16-QAM, a quad LordCore system executing LORD has a better performance than the two-way algorithm in [16], providing hundreds of times better energy efficiency.

On 2×2 64-QAM, a quad LordCore executing LORD has about 3× performance and about 1000 times the energy efficiency compared to the two-way algorithm on the GPU. On 4×4 16-QAM, the one-way algorithm in [16] provides about 300× lower energy efficiency than LordCore executing MMSE. The nonprogrammable fully hardware-based MIMO

TABLE IV
COMPARISON OF MIMO DETECTORS

system	algorithm	ntx / nrx	modulation	throughput	energy/bit	energy/bit scaled(*)	performance/per area scaled(*)
				[Mbps]	[nJ]	[nJ]	[Mpbs/mm^2]
Quadro FX 1700 [14]	LORD			17	2500.	600.	0.8
ASIP cluster [24]	nML			84	0.1	0.1	1714
Tesla C1060 [15]	MTT			270	696.	251.	2.2
GeForce 560 Ti [16]	2-way			402	522.	282.	2.5
Proposed (single core)	LORD	2 / 2	16-QAM	125	0.6	0.6	
Proposed (dual-core)	LORD			241	0.6	0.6	142(***)
Proposed (quad-core)	LORD			472	0.6	0.6	189(**)
LORD-I [12]	LCS-LORD			240	0.2	0.07	774
LORD-II [12]	LCS-LORD			164	0.09	0.04	1228
GeForce 560 Ti [16]	1-way			834	252.	136.	5.1
Proposed (single core)	MMSE	2 / 2	16-QAM	427	0.2	0.2	
Proposed (dual-core)	MMSE			746	0.2	0.2	439(***)
GeForce 560 Ti [16]	1-way			184	1144.	615.	1.1
CGRA[27]	MMSE			480	0.4	0.1	1848
ASIP cluster [24]	SIC	2 / 2	64-QAM	192	0.06	0.04	3918
Tesla C1060 [15]	MTT			44	4277.	1543.	0.4
Proposed (single core)	MMSE			549	0.1	0.1	
GeForce 560 Ti [16]	2-way			92	2273.	1231.	0.6
ASIP cluster [24]	nML			24	0.5	0.4	489
Proposed (single core)	LORD	2 / 2	64-QAM	74	0.9	0.9	
Proposed (dual-core)	LORD			146	0.9	0.9	86(***)
Proposed (quad-core)	LORD			286	0.9	0.9	114(**)
GeForce 560 Ti [16]	1-way			783	268.	145.	4.8
Proposed (single core)	MMSE	4 / 4	16-QAM	119	0.5	0.5	
Proposed (quad-core)	MMSE			454	0.5	0.5	182(**)
GeForce 560 Ti [16]	1-way			231	910.	490.	1.4
Fixed-function HW[10]	MMSE			757	0.3	0.05	5214
Fixed-function HW[11]	k-best			1044	0.1	0.05	
Fixed-function HW[13]	mod. j-best			13300	0.09	0.02	
ASIP cluster [24]	SIC SO			125	0.1	0.07	2551
napCore [25]	MMSE	4 / 4	64-QAM	139	1.3	0.2	2872
Tomahawk BBPM [26]	MMSE			168	2.1	1.9	240
CGRA [27]	MMSE			600	0.4	0.2	2310
Proposed (single core)	MMSE			172	0.4	0.4	
Proposed (quad-core)	MMSE			639	0.4	0.4	256(**)

Most of the energy-efficiency numbers are based on maximum official power consumption values (TDP) or inter-/extrapolated values. (*) Scaled to 28nm, 1.0V. (**) Includes memories. (***) Includes full 448 kiB of global memory.

TABLE V
HARDWARE CHARACTERISTICS OF THE MIMO DETECTORS USED IN THE COMPARISON IN TABLE IV

system	clock rate [MHz]	mfg process [nm]	power consumption [mW]	die size [mm^2]	no. SIMD/SIMT lanes $\times$ data type	programmability
Quadro FX 1700 [14]	920	80	42000*	169.0	32×32b fp	high level(CUDA)
Tesla C1060 [15]	1296	55	188000*	470.0	240×32b fp	high level(CUDA)
GeForce 560 Ti [16]	1645	40	170000*	332.0	336×32b fp	high level(CUDA)
ASIP cluster [24]	800	40	12**	0.1	16×16b fxp	low-level
napCore [25]	400	90	143	0.5	8×19b fp	low-level
Tomahawk BBPM[26]	667/571	28	345	0.7		yes
CGRA [27]	400	65	288	1.4	20×23b fxp	low-level
Fixed-function HW[10]	568	90	189	1.5	8×15b fxp	none
Fixed-function HW[11]	435	65	59			none
Fixed-function HW[13]	556	130				none
LORD-1 HW [12]	80	65	38	0.72		none
LORD-1 HW [12]	80	65	14	0.31		none
Proposed (quad-core)	900	28	269***	2.5	128×16b fp	high-level(OpenCL)

(*) Thermal Design Power. (**) Single core only, without DMA transfer power. (***) Complete design with memory power.

detectors [10], [11], [12] outperform slightly the LordCore in energy efficiency, while the performance is almost equal with a quad LordCore system. However, it should be noted that these use older and less power-efficient manufacturing process, and with a comparable manufacturing process, the difference would be greater. The newest fixed function hardware MIMO detector [13] outperforms Lordcore greatly on both performance and power efficiency, even though it is synthesized to much older 130-nm technology. All these fixed-function MIMO detector hardware blocks can, however, run only single MIMO detection algorithm.

Compared to the napCore [25], LordCore shows both better programmability, performance per core, and energy efficiency, but the proposed system is synthesized for smaller manufacturing technology, and with similar manufacturing technology, napCore would consume less energy per bit. However, it is also to be noted that the power numbers for the napCore exclude the LLR calculations while those are included in the power numbers for the proposed system. Tomahawk [26] with the same manufacturing process shows $5\times$ lower energy efficiency than LordCore. Both have a similar area efficiency.

In 2×2 16-QAM mode, LordCore outperforms the CGRA-based processor in [27] both in terms of throughput and energy efficiency, though with similar manufacturing technology, their energy efficiency would be in the same class. However, in 4×4 16-QAM, the CGRA processor reaches similar class performance and energy efficiency while being manufactured with a larger manufacturing process meaning that it would be faster and more energy efficient on a similar manufacturing process. However, their CGRA requires careful manual programming for obtaining the full performance.

The ASIP cluster in [24] is a hierarchical design consisting of multiple 512b clusters. Each cluster contains two engines, which are based on multiple slices. In architectural perspective, one cluster reminds a core in our design, but the cluster is smaller in terms of transistor count than the proposed cores. The ASIP cluster was synthesized on a 40-nm technology. A cluster consumes 12 mW at 800-MHz clock rate, which means energy efficiency of 500 pJ/bit with near-ML algorithm on 2×2 in 64-QAM mode. On 28-nm technology, this would become about 350 pJ/bit. Our design has efficiency of 906 pJ/bit, but this also includes the channel preprocessing part which their system does not perform. In the same mode, [24] with the SIC algorithm reaches 63 pJ/bit, which would become about 44 pJ/Bit on comparable 28-nm technology. However, this is without the signal preprocessing part. On the same mode, our design provides 122 pJ/bit with MMSE algorithm, including all necessary channel preprocessing. In the 4×4 64-QAM mode, the ASIP cluster achieves 96 pJ/bit, which would become about 67 pJ/bit with comparable 28-nm technology. Our system achieves only 367 pJ/bit, but also contains the channel preprocessing part. The energy efficiency of our design is lower but it should be noted that while the power numbers for both the designs contain also memories, there is a slight difference: in [24], an external direct memory access controller is assumed to transfer data to private memories while, in our design, the data are assumed to be found from a global memory and transferred to private memories under software control, i.e., the power consumption due to this transfer is included in our numbers. In general, the traffic to the global memory represents 27% of the total power consumption in our design. Although taking into account the fact that the power numbers in [24] exclude data transfers to private memories, and in our design, such transfers take 27% of the total power, and they also exclude the channel preprocessing part of the detection, the difference is not that great, when taking into account that our design is much more flexible; it has only few dedicated function units and it is programmed with high abstraction-level OpenCL language, while the design in [24] is highly specialized with limited instruction set and manually programmed with assembly language.

VI. Conclusion

In this paper, a customized processor and implementations for MIMO detector algorithms LORD and MMSE were presented. The proposed architecture provides three orders of magnitude better performance/power ratio than commercial GPU-based solutions, demonstrating the feasibility of the approach. Compared to other highly specialized SDR solutions, the proposed solution consumes slightly more power and area, while the added programmer productivity via floating-point arithmetic and OpenCL-programming are remarkable. Fixed-function ASIC implementations are clearly always the most energy-efficient means for implementing radio functionalities, but the presented results show that the energy penalty paid by programmability can be made very small.

References

[1] B. A. Bjerke, "LTE-advanced and the evolution of LTE deployments," *IEEE Wireless Commun.*, vol. 18, no. 5, pp. 4–5, Oct. 2011.

[2] B. P. Crow, I. Widjaja, J. G. Kim, and P. T. Sakai, "IEEE 802.11 wireless local area networks," *IEEE Commun. Mag.*, vol. 35, no. 9, pp. 116–126, Sep. 1997.

[3] U. Reimers, "DVB-T: The COFDM-based system for terrestrial television," *Electron. Commun. Eng. J.*, vol. 9, no. 1, pp. 28–32, Feb. 1997.

[4] G. Faria, J. A. Henriksson, E. Stare, and P. Talmola, "DVB-H: Digital broadcast services to handheld devices," *Proc. IEEE*, vol. 94, no. 1, pp. 194–209, Jan. 2006.

[5] G. Smith, "Updates of the ITRS design cost and power models," in *Proc. IEEE Int. Conf. Comput. Design*, Seoul, South Korea, Oct. 2014, pp. 161–165.

[6] U. Madhow and M. L. Honig, "MMSE interference suppression for direct-sequence spread-spectrum CDMA," *IEEE Trans. Commun.*, vol. 42, no. 12, pp. 3178–3188, Dec. 1994.

[7] M. Siti and M. Fitz, "A novel soft-output layered orthogonal lattice detector for multiple antenna communications," in *Proc. IEEE Int. Conf. Commun.*, Istanbul, Turkey, Jun. 2006, pp. 1686–1691.

[8] *The OpenCL Specification*, Standard Rev. v1.2r15, Khronos Group, Nov. 2012. [Online]. Available: http://www.khronos.org/registry/cl/

[9] H. Corporaal, *Microprocessor Architectures: From VLIW to TTA*. Chichester, U.K.: Wiley, 1997.

[10] C. Studer, S. Fateh, and D. Seethaler, "ASIC implementation of soft-input soft-output MIMO detection using MMSE parallel interference cancellation," *IEEE J. Solid-State Circuits*, vol. 46, no. 7, pp. 1754–1765, Jul. 2011.

[11] J. Cheng, J. Zheng, X. Zhou, and L. Zhang, "Implementation of a configurable MIMO detector with complex K-best algorithm," in *Proc. IEEE Int. Conf. ASIC*, Shenzhen, China, Oct. 2013, pp. 1–4.

[12] T. Cupaiuolo, M. Siti, and A. Tomasoni, "Low-complexity high throughput VLSI architecture of soft-output ML MIMO detector," in *Proc. Conf. Design Autom. Test Eur.*, Dresden, Germany, Mar. 2010, pp. 1396–1401.

[13] M. Mahdavi and M. Shabany, "A 13 Gbps, 0.13 μm CMOS, multiplication-free MIMO detector," *J. Signal Process. Syst.*, vol. 88, no. 3, pp. 273–285, Sep. 2017.

[14] T. Nyländen, J. Janhunen, O. Silvén, and M. Juntti, "A GPU implementation for two MIMO-OFDM detectors," in *Proc. Int. Conf. Embedded Comput Syst., Arch. Modeling Simulation*, Samos, Greece, Jul. 2010, pp. 293–300.

[15] M. Wu, Y. Sun, S. Gupta, and J. R. Cavallaro, "Implementation of a high throughput soft MIMO detector on GPU," *J. Signal Process. Syst.*, vol. 64, pp. 123–136, Jul. 2011.

[16] M. Wu, B. Yin, and J. Cavallaro, "Flexible N-way MIMO detector on GPU," in *Proc. IEEE Workshop Signal Process. Syst.*, Quebec City, QC, Canada, Oct. 2012, pp. 318–323.

[17] Y. Lin *et al.*, "SODA: A low-power architecture for software radio," *ACM SIGARCH Comput. Arch. News*, vol. 34, no. 2, pp. 89–101, 2006.

[18] M. Woh *et al.*, "From SODA to scotch: The evolution of a wireless baseband processor," in *Proc. IEEE/ACM Int. Symp. Microarchitecture*, Lake Como, Italy, Nov. 2008, pp. 152–163.

[19] C. H. Chou, A. Severance, A. D. Brant, Z. Liu, S. Sant, and G. G. Lemieux, “VEGAS: Soft vector processor with scratchpad memory,” in *Proc. ACM/SIGDA Int. Symp. Field Prog. Gate Arrays*, Monterey, CA, USA, Feb./Mar. 2011, pp. 15–24.

[20] M. Woh *et al.*, “AnySP: Anytime anywhere anyway signal processing,” *IEEE Micro*, vol. 30, no. 1, pp. 81–91, Jan. 2010.

[21] L. Waeijen, D. She, H. Corporaal, and Y. He, “SIMD made explicit,” in *Proc. Int. Conf. Embedded Comput. Syst., Arch. Modeling Simulation*, Samos, Greece, Jul. 2013, pp. 330–337.

[22] D. C.-W. Chang *et al.*, “Parallel architecture core (PAC)—The first multicore application processor SoC in Taiwan. Part I: Hardware architecture & software development tools,” *J. Signal Process. Syst.*, vol. 62, no. 3, pp. 373–382, Mar. 2011.

[23] J. Zalamea, J. Llosa, E. Ayguadé, and M. Valero, “Two-level hierarchical register file organization for VLIW processors,” in *Proc. Annu. ACM/IEEE Int. Symp. Microarchitecture*, Monterey, CA, USA, Dec. 2000, pp. 137–146.

[24] R. Fasthuber, P. Raghavan, L. V. D. Perre, and F. Catthoor, “A scalable MIMO detector processor with near-ASIC energy efficiency,” *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 23, no. 10, pp. 1973–1986, Oct. 2015.

[25] D. Guenther, R. Leupers, and G. Ascheid, “Efficiency enablers of lightweight SDR for MIMO baseband processing,” *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 24, no. 2, pp. 567–577, Feb. 2016.

[26] S. Haas *et al.*, “A heterogeneous SDR MPSoC in 28 nm CMOS for low-latency wireless applications,” in *Proc. Design Autom. Conf.*, Austin, TX, USA, 2017, pp. 47:1–47:6.

[27] X. Chen, A. Minwegen, S. B. Hussain, A. Chattopadhyay, G. Ascheid, and R. Leupers, “Flexible, efficient multimode MIMO detection by using reconfigurable ASIP,” *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 23, no. 10, pp. 2173–2186, Oct. 2015.

[28] B. Mei, A. Lambrechts, J. Y. Mignolet, D. Verkest, and R. Lauwereins, “Architecture exploration for a reconfigurable architecture template,” *IEEE Design Test Comput.*, vol. 22, no. 2, pp. 90–101, Mar. 2005.

[29] U. Ahmad *et al.*, “Exploration of lattice reduction aided soft-output MIMO detection on a DLP/ILP baseband processor,” *IEEE Trans. Signal Process.*, vol. 61, no. 23, pp. 5878–5892, Dec. 2013.

[30] B. Bougard, B. De Sutter, D. Verkest, L. Van der Perre, and R. Lauwereins, “A coarse-grained array accelerator for software-defined radio baseband processing,” *IEEE Micro*, vol. 28, no. 4, pp. 41–50, Jul./Aug. 2008.

[31] S. Shahabuddin, J. Janhunen, Z. Khan, M. Juntti, and A. Ghazi, “A customized lattice reduction multiprocessor for MIMO detection,” in *Proc. IEEE Int. Symp. Circuits Syst.*, Lisbon, Portugal, May 2015, pp. 2976–2979.

[32] T. Nyländen, H. Kultala, I. Hautala, J. Boutellier, J. Hannuksela, and O. Silvén, “Programmable data parallel accelerator for mobile computer vision,” in *Proc. IEEE Global Conf. Signal Inf. Process.*, Orlando, FL, USA, Dec. 2015, pp. 624–628.

[33] J. Hoogerbrugge and H. Corporaal, “Register file port requirements of transport triggered architectures,” in *Proc. Annu. Int. Symp. Microarchitecture*, San Jose, CA, USA, Nov./Dec. 1994, pp. 191–195.

[34] J. Janhunen, T. Pitkänen, O. Silvén, and M. Juntti, “Fixed- and floating-point processor comparison for MIMO-OFDM detector,” *IEEE J. Sel. Topics Signal Process.*, vol. 5, no. 8, pp. 1588–1598, Dec. 2011.

[35] M. S. Peercy, M. Olano, J. Airey, and P. J. Ungar, “Interactive multi-pass programmable shading,” in *Proc. Ann. Conf. Comput. Graph. Interact. Tech.*, New Orleans, LA, USA, Jul. 2000, pp. 425–432.

[36] *IEEE Standard for Floating-Point Arithmetic*, IEEE Standard 754-2008, Aug. 2008.

[37] S. M. Robinson, “A short proof of Cramer’s rule,” *Math. Mag.*, vol. 43, no. 2, pp. 94–95, 1970.

[38] C. Lomont, “Fast inverse square root,” Purdue Univ., West Lafayette, IN, USA, Tech. Rep. 32, Feb. 2003.

[39] H. Corporaal and M. Arnold, “Using transport triggered architectures for embedded processor design,” *Integr. Comput.-Aided Eng.*, vol. 5, no. 1, pp. 19–38, 1998.

[40] V. Guzma, P. Jääskeläinen, P. Kellomäki, and J. Takala, “Impact of software bypassing on instruction level parallelism and register file traffic,” in *Embedded Computer Systems: Architectures, Modeling, and Simulation* (Lecture Notes in Computer Science), vol. 5114, M. Bereković, N. Dimopoulos, and S. Wong, Eds. Berlin, Germany: Springer, 2008, pp. 23–32.

[41] H. Kultala, J. Multanen, P. Jääskeläinen, T. Viitanen, and J. Takala, “Impact of operand sharing to the processor energy efficiency,” in *Proc. Int. Symp. Comput. Arch. Digit. Systs*, Tehran, Iran, Oct. 2015, pp. 1–6.

[42] P. Jääskeläinen, T. Viitanen, J. Takala, and H. Berg, “HW/SW co-design toolset for customization of exposed datapath processors,” in *Computing Platforms for Software-Defined Radio*, W. Hussain, J. Nurmi, J. Isoaho, and F. Garzia, Eds. Cham, Switzerland: Springer, 2017, pp. 147–164.

[43] P. Jääskeläinen, C. de La Lama, P. Huerta, and J. Takala, “OpenCL-based design methodology for application-specific processors,” in *Proc. Int. Conf. Embedded Comput. Syst.*, Samos, Greece, Jul. 2010, pp. 223–230.

[44] P. Jääskeläinen, C. S. de La Lama, E. Schnetter, K. Raiskila, J. Takala, and H. Berg, “pocl: A performance-portable OpenCL implementation,” *Int. J. Parallel Program.*, vol. 43, no. 5, pp. 752–785, 2015.

[45] C. Lattner and V. Adve, “LLVM: A compilation framework for lifelong program analysis & transformation,” in *Proc. Int. Symp. Code Gener. Optim.*, Palo Alto, CA, USA, Mar. 2004, pp. 75–87.

[46] J.-J. Li, C.-B. Kuan, T.-Y. Wu, and J. K. Lee, “Enabling an OpenCL compiler for embedded multicore DSP systems,” in *Proc. Int. Conf. Par. Process. Workshop*, Pittsburgh, PA, USA, Sep. 2012, pp. 545–552.

[47] H. O. Kultala, T. T. Viitanen, P. Jääskeläinen, and J. H. Takala, “Aggressively bypassing list scheduler for transport triggered architectures,” in *Proc. Int. Conf. Embedded Comput. Syst., Arch. Modeling Simulation*, Samos, Greece, Jul. 2016, pp. 253–260.

[48] T. Viitanen, H. Kultala, P. Jääskeläinen, and J. Takala, “Heuristics for greedy transport triggered architecture interconnect exploration,” in *Proc. Int. Conf. Compil. Arch. Synthesis Embedded Syst.*, New Delhi, India, vol. 2, Oct. 2014, pp. 1–7.

[49] P. Jääskeläinen, E. Salminen, O. Esko, and J. Takala, “Customizable datapath integrated lock unit,” in *Proc. Int. Symp. Syst.-Chip*, Tampere, Finland, Oct./Nov. 2011, pp. 29–33.

[50] J. Multanen *et al.*, “Power optimizations for transport triggered SIMD processors,” in *Proc. Int. Conf. Embedded Comput. Syst., Arch. Modeling Simulation*, Samos, Greece, Jul. 2015, pp. 303–309.

[51] C. Berrou, A. Glavieux, and P. Thitimajshima, “Near Shannon limit error-correcting coding and decoding: Turbo-codes. 1,” in *Proc. IEEE Int. Conf. Commun. (ICC)*, Geneva, Switzerland, vol. 2, May 1993, pp. 1064–1070.

Heikki Kultala received the M.Sc. degree in computer science from the Tampere University of Technology, Tampere, Finland, in 2010, where he is currently working toward the D.Sc. degree.

Since 2005, he has been involved in exposed datapath processor architecture design and programming. His current research interests include compiler backend techniques, low-level code optimizations, and processor architectures.

Timo Viitanen received the M.Sc. and D.Sc. degrees in embedded systems from the Tampere University of Technology, Tampere, Finland, in 2013 and 2018, respectively. His dissertation was focused on Hardware Accelerators for Animated Ray Tracing.

His current research interests include computer architecture and computer graphics.

Heikki Berg received the M.Sc. degree in telecommunications from the University of Oulu, Oulu, Finland, in 1998.

He was at Nokia, where he was involved in research and implementation of signal processing solutions for wireless radio systems and for audio, and his team codesigned several practical application-specific processor prototypes. Since 2018, he has been at Epec Oy, Seinäjoki, Finland, where he was involved in autonomous moving machines.

Pekka Jääskeläinen is currently an Assistant Professor at Tampere University, Tampere, Finland. In addition to publication activities, he is the Lead Developer of multiple heterogeneous computing-related open-source projects at Tampere University.

His current research interests include methods and tools to ease design and programming of diverse heterogeneous platforms, and reducing the energy consumption of software-based control hardware.

Joonas Multanen received the M.Sc. degree in electrical engineering from the Tampere University of Technology, Tampere, Finland, in 2015, where he is currently working toward the D.Sc. degree.

His current research interests include energy-efficient computer architectures, emerging memory technologies, and computer graphics.

Mikko Kokkonen received the M.Sc. (Hons.) and Lic.Tech. degrees from the Helsinki University of Technology, Espoo, Finland, in 1991 and 1999, respectively.

Since 1993, he has been at Nokia, Espoo. From 2011 to 2013, he was at Renesas Mobile, Espoo. He is involved in software-defined implementations of base station algorithms. His current research interests include design and implementation of radio systems and algorithms.

Kalle Raiskila received the M.Sc.E.E. degree from the Helsinki University of Technology, Espoo, Finland, in 2006.

He is currently an Embedded Systems Software Specialist at Senseg, Espoo. His current research interests include haptic feedback systems.

Tommi Zetterman received the M.Sc. degree in computer science from the Helsinki University of Technology, Espoo, Finland, in 1996. From 1994 to 2013, he was at the Nokia Research Center, Helsinki, Finland, where he was involved in computer architectures, low-power ASIC design techniques, physical layer radio computation, and software-defined radio. From 2014 to 2017, he was at Nokia Technologies, where he was a Principal Engineer. Since 2018, he has been at Nokia Networks, Espoo, where he is currently a Senior SoC Specialist.

Jarmo Takala (SM'02) received the M.Sc. (Hons.) and D.Sc.(Tech.) degrees from the Tampere University of Technology (TUT), Tampere, Finland, in 1987 and 1999, respectively.

In 1992, he joined VTT-Automation, Tampere. In 1995, he joined the Nokia Research Center, Tampere. In 1996, he joined TUT. He is a Professor of Computer Engineering at Tampere University, Tampere.

Mr. Takala was the Chair of the IEEE Signal Processing Society's Design and Implementation of Signal Processing Systems Technical Committee from 2012 to 2013. From 2007 to 2011, he was an Associate Editor and an Area Editor for the IEEE TRANSACTIONS SIGNAL PROCESS.