Energy-efficiency of floating-point and fixed-point SIMD cores for MIMO processing systems

D. Guenther, A. Bytyn, R. Leupers, G. Ascheid
{guenther, bytyn, leupers, ascheid}@ice.rwth-aachen.de

Abstract - For computational tasks with regular data access patterns, e.g. vector arithmetic, single instruction multiple data (SIMD) processor cores present a viable alternative to application-specific integrated circuits (ASICs). Even though the additional flexibility of a programmable core comes at the expense of reduced area- and energy efficiency, this price is potentially worth paying in application domains with a multitude of standards and use cases as it is the case in the domain of wireless communications. The high dynamic range of values that occurs in multi-antenna wireless baseband processing calls for the use of numerical stabilization measures like QR factorization and scaling when operating on a fixed-point processor core. A floating-point core on the other hand covers a wider dynamic range, rendering such measures unnecessary at the price of increased power consumption. This work compares a floating-point and a fixed-point SIMD core in a case study of linear MIMO detection. After evaluating the numerical precision requirements, the achievable throughput of both cores is compared along with the maximum achievable area- and energy efficiency for several use cases, resulting in an assessment as to which number format is best suited for which use case.

I. Introduction

Today's versatile wireless mobile communication landscape includes a multitude of communication standards like IEEE 802.11a/b/g/n/ac [1], GSM, UMTS, LTE, and LTE Advanced [2]. This variety has motivated the use of software defined radios (SDRs), programmable platforms on which new standards can be implemented in software, which increases flexibility and enables faster deployment. The price to pay is a decreased efficiency in terms of energy and area when compared to tailored application-specific integrated circuits (ASICs). However, as we already demonstrated in [3], this price is less than the commonly mentioned orders of magnitude [4], if the SDR platform comprises well designed application-specific instruction set processors (ASIPs). In case of [3], the efficiency-enabling ASIP is a floating-point single instruction multiple data (SIMD) processor core. It features a tailored instruction set for complex vector arithmetic, support for versatile data access patterns as well as numerically aware processing capabilities, allowing the programmer to adapt the mantissa width to the application's precision requirements.

The choice of floating-point arithmetic was initially motivated by the high dynamic ranges that occur particularly in iterative MIMO detection algorithms in baseband processing. The floating-point number format allows the use of less complex algorithms without explicit numerical stabilization (e.g. by matrix factorization), which decreases implementation effort and increases throughput. On the other hand, hardware for floating-point arithmetic is more complex than its fixed-point counterpart, indicating that the slower fixed-point implementation might also consume less power. This raises the question as to how both implementations perform in terms of energy efficiency. Will the shortened runtime of the floating-point implementation over-compensate the increased power consumption of the complex arithmetic units or will the fixed-point implementation, despite the longer runtime, still be the more energy-efficient solution? This question is of paramount importance, especially in the domain of battery powered devices.

We aim to investigate this question by first designing a comparable fixed-point counterpart to our floating-point SIMD processor core introduced in [3]. Then both cores are compared in the exemplary application domain of linear MIMO detection, covering both open-loop (non-iterative) as well as iterative detection. A variety of numerical stabilization algorithms exist to stabilize the required matrix inversion for fixed-point arithmetic. The available choices and their implementation on the fixed-point core are presented with respect to runtime and precision requirements, which ultimately relate to energy consumption. Based on topographical synthesis results for the required precision, the energy consumption is then evaluated and compared to the floating-point core in [3]. In a more in-depth analysis, we study the energy consumption of distinct sections within linear detection to get a more detailed understanding of the efficiency of each number format for different use cases.

The remainder of this paper is structured as follows: Section II introduces the architectures of the two SIMD processor cores. Section III provides a quick overview of linear MIMO detection as the basis for our case study. Section IV presents the implementation of aforementioned detection algorithms on the two cores and compares throughput as well as energy efficiency for the algorithms as a whole as well as for different fractions. Section V concludes the paper.

II. Architecture overview

This section gives an overview of the two investigated ASIP architectures, outlining their common features as well as their specific properties.

A. Common features

Both cores are fully programmable via a program memory (*PMEM*) that is triggered in the pre-fetch stage (*PFE*) and read in the fetch stage (*FE*). They are implemented as 4-way SIMD architectures with native support for complex-valued vector arithmetic. Their arithmetic logical unit (ALU) is designed to match the structure of most vector arithmetic operations, starting with a series of multipliers followed by a reconfigurable adder tree, which can, for example, perform the reduction of the result of a component-wise vector-vector multiplication to a single scalar. A scalar and a vector register file are available

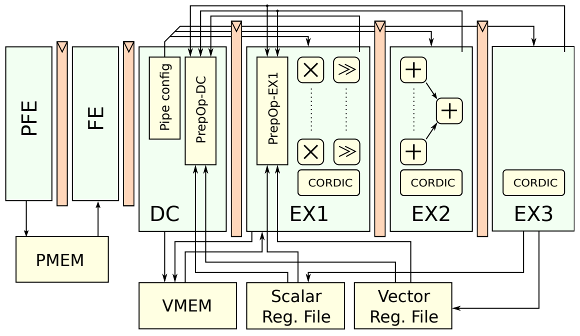

Fig. 1. Fixed-point SIMD processor core pipeline overview

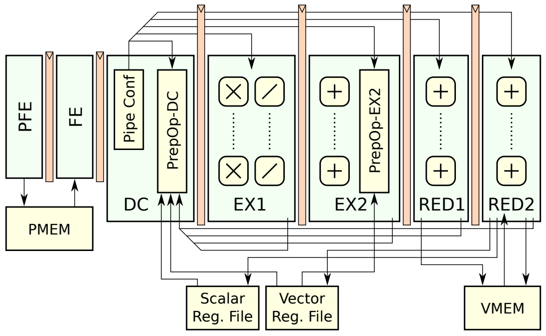

Fig. 2. Floating-point SIMD processor core pipeline overview

for data loading and storing, enabling a versatile set of operand acquisition schemes, e.g. for scalar-vector multiplications. The operand width can be reduced by an adaptive bitmask applied to a set of least significant bits (LSBs) of each fixed-point word or mantissa in case of the floating-point variant. This technique is used to reduce energy consumption, e.g. when performing detection for a less dense symbol constellation with reduced numerical precision requirements. Masking is performed after operand loading from the register files as well as after each pipeline stage of the ALU. Operands can be loaded from the register files or bypassed from the pipeline from the *PrepOp* units where the one in *DC* loads the input to the multipliers while the second one fetches further inputs to the adder tree (e.g. for multiply-accumulate operations). Data storage is provided by a vector memory (*VMEM*), where the memory width corresponds to the width of one vector operand.

B. Fixed-point variant

Fig. 1 depicts the pipeline structure of the fixed-point processor core. Depending on whether or not the CORDIC unit, that spans across three stages, is included, the core comprises five or six pipeline stages. The CORDIC unit can be used to explore matrix factorization methods that rely on vector rotation (e.g. Givens rotation). The ALU contains three stages (*EX1*, *EX2*, *EX3*), where *EX1* contains the complex valued multipliers as well as the reconfigurable backshifters, which can be used to support fixed-point number formats with a variable decimal point position. Stage *EX2* contains the configurable adder tree, which, as opposed to the floating-point design, fits into one pipeline stage due to the reduced hardware complexity of fixed-point adders in contrast to floating-point adders. The ALU also implements scalar inversions as well as inverse square root calculations based on the Newton-Raphson algorithm. No additional hardware is required to map these operations. The preexisting vector data path is reused, instead.

C. Floating-point variant

The floating-point core as depicted in Fig. 2 contains seven pipeline stages with an ALU spanning over four stages. The increased length, as opposed to the fixed-point variant, can be explained by the increased hardware complexity of floating-point adders. As a result, the complex-valued scalar multiplications are spread across stages *EX1* and *EX2*, where real-valued multiplications are executed in *EX1*, followed by additions in *EX2* to form the actual complex valued results. The reconfigurable adder tree requires one stage per adder-depth, resulting in two reduction stages (*RED1*, *RED2*) for the case of 4-way SIMD. The *EX1* stage also contains a set of parallel dedicated reciprocal units implementing the Newton-Raphson algorithm for scalar inversion. As opposed to the fixed-point variant, multiplications and additions of the existing data path are not reused to map the inversion algorithm, since the normalized floating-point mantissa allows a simplified scalar inversion without considering a number's exponent.

III. Linear MIMO detection algorithms

The following section gives a brief overview of linear MIMO detection algorithms. For more detailed information, the interested reader is referred to [5]. We consider a multi-antenna transmission with N_t transmit and N_r receive antennas via a Rayleigh fading channel superimposed by white Gaussian noise. Thus, the channel is modeled as

$$\mathbf{y} = \mathbf{H}\mathbf{x} + \mathbf{n}, \tag{1}$$

with receive vector $\mathbf{y}$ of dimension N_r, transmit vector $\mathbf{x}$ of dimension N_t, additive noise vector $\mathbf{n}$ of dimension N_r as well as channel matrix $\mathbf{H}$ of dimension $N_r \times N_t$.

A. Open-loop linear MMSE MIMO detection

To estimate the originally transmitted symbol vector $\mathbf{x}$ from the received symbol vector $\mathbf{y}$, while minimizing the mean square error among the estimate and $\mathbf{x}$ itself, $\mathbf{y}$ is multiplied by an equalizer matrix given as

$$\mathbf{G} = \left(\hat{\mathbf{H}}^H \hat{\mathbf{H}} + N_0 \mathbf{I}_{N_t}\right)^{-1} \hat{\mathbf{H}}^H, \tag{2}$$

with estimated channel matrix $\hat{\mathbf{H}}$, identity matrix $\mathbf{I}_{N_t}$ of dimension $N_t \times N_t$ and noise spectral density N_0. In a next step the estimate $\hat{\mathbf{x}}$ is transferred to a log-likelihood representation for each bit i of each antenna element k of the symbol vector according to

$$L(b_{k,i}) \approx \rho_k \left(\min_{s \in A_i^0} |\hat{x}_k - s|^2 - \min_{s \in A_i^1} |\hat{x}_k - s|^2 \right), \tag{3}$$

where A_i^0 denotes the constellation symbols with a *zero* at position i of their bitwise representation and A_i^1 denoting the constellation symbols with a *one* at the same position respectively. Based on this, the signal-to-interference-plus-noise ratio (SINR) ρ_k of the k-th antenna stream is derived by

$$\rho_k \approx \frac{1}{N_0 \left[\left(\hat{\mathbf{H}}^H \hat{\mathbf{H}} + N_0 \mathbf{I}_{N_t} \right)^{-1} \right]_{k,k}} - 1. \tag{4}$$

B. Iterative linear MMSE MIMO detection

As an addition to open-loop detection, iterative detection also considers priori information from previous iterations, e.g. in the shape of posteriori LLRs. The concept is described in [6] and algorithmically reformulated in [7], resulting in the parallel interference cancellation (MMSE-PIC) algorithm as described in the following.

Using an estimate $\mathbf{s}$ of the transmitted symbol vector $\mathbf{x}$ based on posteriori LLRs, inter-antenna interference is reduced by calculating vectors

$$\hat{\mathbf{y}}_k = \mathbf{y} - \sum_{j \neq k} \mathbf{h}_j s_j, \tag{5}$$

for $k \in \{1..N_t\}$. Each of these vectors is an estimate for what would have been received if only the k-th antenna had been transmitting.

A further MMSE filter is applied based on the matrix

$$\mathbf{W}^H = \left(\hat{\mathbf{H}}^H \hat{\mathbf{H}} \mathbf{\Lambda} + N_0 \mathbf{I} \right)^{-1} \hat{\mathbf{H}}^H, \tag{6}$$

with diagonal matrix $\mathbf{\Lambda}$ of dimension $N_t \times N_t$ containing the element-wise symbol variances of $\mathbf{s}$. By expressing the rows of $\mathbf{W}^H$ as $[\mathbf{w}_1 \ .. \ \mathbf{w}_{N_t}]^H$, the filtering is given as

$$\hat{\mathbf{x}}_k = \frac{\mathbf{w}_k^H \hat{\mathbf{y}}_k}{\mathbf{w}_k^H \mathbf{h}_k}. \tag{7}$$

The new LLRs for the i-th bit within the k-th element of vector $\hat{\mathbf{x}}$ are calculated as in (3) using the altered SINR

$$\rho_k = \frac{\mathbf{w}_k^H \mathbf{h}_k}{1 - \mathbf{\Lambda}_{kk} \mathbf{w}_k^H \mathbf{h}_k}. \tag{8}$$

IV. Software implementation

The following describes the software implementation of aforementioned algorithms on the two processor cores and discusses implementation alternatives along with their resulting cycle count and precision requirement, which varies depending on the used constellation (e.g. 4QAM vs. 64QAM). In terms of numerical stability, the inversions in (2) and (6) are the most critical operations within the detection process, so special emphasis is put on this aspect.

A. Fixed-point implementation

Due to the small dynamic range of fixed-point arithmetic, numerical stabilization is required to calculate (2) and (6). The following gives a brief overview of the considered matrix factorization methods. A more detailed description is provided by [8].

1) QR factorization: decomposes a matrix $\mathbf{A}$ of dimension $M \times N$ into a product of two matrices $\mathbf{Q}$ and $\mathbf{R}$, where $\mathbf{Q}$ is a unitary matrix of dimension $M \times N$ and $\mathbf{R}$ is a triangular matrix of dimension $N \times N$. These properties allow to rewrite e.g. the left side matrix inverse in a numerically more stable manner

$$\begin{aligned} \mathbf{A} &= \mathbf{QR} && (9) \\ \mathbf{A}^{-1} &= \mathbf{R}^{-1} \mathbf{Q}^H, && (10) \end{aligned}$$

which only involves the inverse of triangular matrix $\mathbf{R}$.

Several algorithms exist to implement the aforementioned decomposition. This work investigates the modified Gram-Schmidt (MGS) algorithm that derives the unitary Q-matrix from a series of vector projections and subtractions and the Givens rotation (GR) algorithm, which is based on vector rotations and hence inherently suitable for a CORDIC based implementation. The same alternatives exist for the regularized factorization.

2) Regularized QR factorization: decomposes a regularized matrix

$$\bar{\mathbf{A}} = \begin{pmatrix} \mathbf{A} \\ c\mathbf{I}_N \end{pmatrix} \tag{11}$$

of dimension $(M + N) \times N$ with scalar c into

$$\bar{\mathbf{A}} = \bar{\mathbf{Q}} \bar{\mathbf{R}} = \begin{pmatrix} \mathbf{Q_a} \\ \mathbf{Q_b} \end{pmatrix} \bar{\mathbf{R}}. \tag{12}$$

Matrix $\mathbf{Q}_a$ is of dimension $M \times N$, $\mathbf{Q}_b$ has dimension $N \times N$ and $\bar{\mathbf{R}}$ covers $N \times N$. Note that by setting $\mathbf{A} = \mathbf{H}$, $c = N_0$, $M = N_r$, and $N = N_t$, the equalizer matrix in (2) can be directly calculated as

$$\mathbf{G} = \frac{1}{N_0} \mathbf{Q}_b \mathbf{Q}_a. \tag{13}$$

However, the prerequisite to this simplification is that the matrix to be inverted is positive semidefinite, which is only the case for open-loop detection.

3) LDLH factorization: decomposes a hermitian, positive definite matrix $\mathbf{A}$ into the product

$$\mathbf{A} = \mathbf{LDL}^H \tag{14}$$

of a lower triangular matrix $\mathbf{L}$, a diagonal matrix $\mathbf{D}$, and the hermitian $\mathbf{L}^H$ of the first matrix. This allows a simplified calculation of the matrix inverse

$$\mathbf{A}^{-1} = \left(\mathbf{L}^{-1} \right)^H \mathbf{D}^{-1} \mathbf{L}^{-1}. \tag{15}$$

Note that the inversions in (15) are particularly easy to calculate, since $\mathbf{L}$ is lower triangular and $\mathbf{D}$ is diagonal, meaning its inverse is a diagonal matrix composed of the reciprocals of the original matrix.

Algorithm	**4QAM**		**16QAM**		**64QAM**	
	ol	it	ol	it	ol	it
Fixed-point						
QR by MGS	-	Q6.12	-	Q6.14	-	Q6.16
QR by GR	-	Q6.10	-	Q6.12	-	Q6.14
Regularized QR by MGS	Q4.10	-	Q4.12	-	Q4.14	-
Regularized QR by GR	Q4.8	-	Q4.10	-	Q4.12	-
LDLH	Q6.14	-	Q6.16	-	Q6.18	-
LU	Q6.14	Q6.14	Q6.16	Q6.16	Q6.18	Q6.18
Floating-point						
DnQ	s1m8e6	s1m6e6	s1m11e6	s1m8e6	s1m12e6	s1m11e6

TABLE I
MAXIMUM PRECISION REQUIREMENTS FOR 4×4 MATRIX INVERSION

4) LU factorization: decomposes matrix $\mathbf{A}$ into a lower- and an upper triangular matrix.

$$\mathbf{A} = \mathbf{LU} \tag{16}$$

For $M \geq N$, $\mathbf{L}$ has dimension $M \times N$ and $\mathbf{U}$ is $N \times N$, and the overall inverse is given by

$$\mathbf{A}^{-1} = \mathbf{U}^{-1}\mathbf{L}^{-1}. \tag{17}$$

Table II summarizes the runtime of the previously introduced algorithms for the decomposition as well as the complete inversion of a matrix $\mathbf{A}$ of dimension 4×4.

Note that energy consumption does not only depend on the runtime but also on the precision requirements of the algorithm under test. Table I lists the maximum requirements for the aforementioned algorithms for a 4QAM, 16QAM, and 64QAM constellation when running an open-loop (ol) or iterative (it) detection. The number of bits is given in Q-notation listing the number of bits before and after the decimal point. Precisions were obtained by extensive Monte Carlo simulations of a physical layer communication based on the IEEE 802.11n standard with a 4×4 antenna setup, where word widths were reduced up to the point just before the reduction starts to compromise the achieved coded bit-error-rate (BER).

B. Floating-point implementation

Due to the high dynamic range covered by floating-point arithmetic, the inversions in (2) and (6) are less critical than in the fixed-point implementation and can be realized by an efficient divide-and-conquer (DnQ) algorithm as presented in [9]. The inversion of a 4×4 matrix $\mathbf{A}$ e.g. can be described based on a series of operations on the 2×2 matrices $\mathbf{a}$, $\mathbf{b}$, $\mathbf{c}$, and $\mathbf{d}$ as in

$$\mathbf{A} = \begin{pmatrix} \mathbf{a} & \mathbf{b} \\ \mathbf{c} & \mathbf{d} \end{pmatrix}. \tag{18}$$

The inverse of $\mathbf{A}$ can then be calculated by

$$\mathbf{A}^{-1} = \begin{pmatrix} \mathbf{a}^{-1} + \mathbf{a}^{-1}\mathbf{bsca}^{-1} & -\mathbf{a}^{-1}\mathbf{bs} \\ \mathbf{sca}^{-1} & \mathbf{s} \end{pmatrix} \tag{19}$$

with $\mathbf{s} = (\mathbf{d} - \mathbf{ca}^{-1}\mathbf{b})^{-1}$.

The precision and runtime requirements of the algorithm are listed in Table I and II, respectively. With respect to precision, s stands for the sign bit and m indicates the number of mantissa bits, while e denotes the number of exponent bits.

C. Discussion

While the LU, LDLH and MGS based algorithms perform at similar cycle-counts, GR requires significantly more runtime, since the CORDIC based matrix triangularization [10] does not exploit the full parallelism offered by a vector processor. Since the regularized algorithm decomposes a matrix with $M + N$ rows instead of just M rows, the runtime for regularized QR decomposition is higher compared to the non-regularized version. The floating-point DnQ algorithm requires the least runtime for inversion, as it does not rely on matrix decomposition. For the same reason, the software development time was significantly shorter. Regarding precision requirements, it can be noted that denser constellations require higher precision on both architectures.

V. SYNTHESIS RESULTS & ENERGY EFFICIENCY

This section presents synthesis results for both processor cores. The core configurations for synthesis are chosen on a per-algorithm basis. Thus, data path widths are set to support a numerically stable execution of the algorithm for all constellations. Fig. 3 shows the hardware complexity in kilo gate equivalents (kGE) as a function of the clock period T_c for the relevant word widths for both architectures.

The points of maximum area efficiency in terms of area-timing (AT) product are summarized in Table III, which lists clock frequency f^A and silicon areas A_c^A and A_{nc}^A for a chip setup with and without CORDIC unit respectively. One sees that the fixed-point implementation achieves significantly lower clock frequencies, due to the longer critical path running through the wider multiplication units. To study energy efficiency, Fig. 4 shows the energy per clock cycle consumed by a high load benchmark. This benchmark calculates a series of matrix-matrix multiplications implemented as inner

Algorithm	**Decomp. [cycles]**	**Inv. [cycles]**
Fixed-point		
QR by MGS	73	89
QR by GR	113	154
Regularized QR by MGS	92	116
Regularized QR by GR	233	288
LDLH	62	106
LU	52	99
Floating-point		
DnQ	-	47

TABLE II
RUNTIME FOR 4×4 MATRIX DECOMPOSITION AND INVERSION

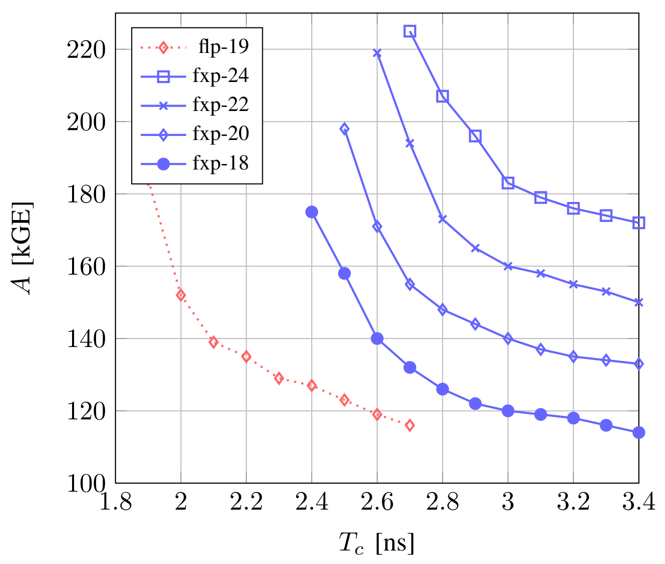

Fig. 3. Area-timing chart for fixed-point and floating-point core

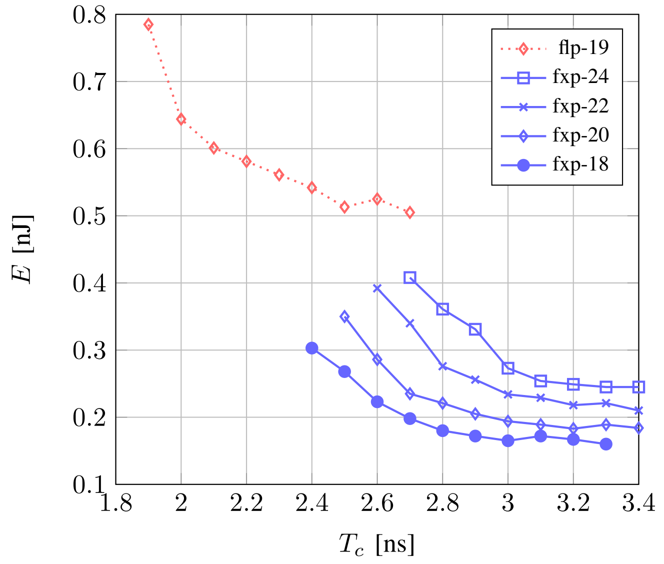

Fig. 4. Energy-timing chart for fixed-point and floating-point core

products and thereby gives a realistic estimate of the energy consumption under maximum load. It is interesting to see that the floating-point processor's energy consumption per cycle is significantly higher than of the fixed-point variant. On the other hand, Table II already indicates that the increased energy consumption per cycle might be compensated by the faster runtime on the floating-point core.

A. *Energy efficiency of fixed-point linear detection*

In the following, the energy consumption of the linear MIMO detection algorithms, when using the matrix inversion algorithms as presented in Section IV, is analyzed. For that purpose, each algorithm is run on the processor configuration with the lowest word width that offers enough precision to ensure a numerically stable execution for 4QAM, 16QAM and 64QAM modulation. The results of the investigation are summarized in Table IV. Average energy E_{avg} is listed per use case along with the fraction of energy τ that can be saved by operand masking, since less dense constellations do not require full precision.

The table allows to compare the energy efficiency of the different fixed-point matrix stabilization methods and also lists the maximum required word width W_c. For open-loop detection, the modified Gram-Schmidt variant performs best, due to the low runtime combined with a low precision requirement. Even though Givens rotation has the lowest precision requirements, the increased runtime due to irregular data access patterns renders the algorithm inefficient for an implementation on a SIMD processor. Note that this finding does not necessarily transfer to ASIC solutions, as demonstrated in [11]. The next best performing fixed-point alternatives are the triangularization based LU and LDLH decompositions, which have similar runtime but higher precision requirements. This picture changes when switching to iterative detection. The higher dynamic range, especially due to the variance term in (6), can be covered by the application of dynamic scaling, which comes at the cost of a slight increase in runtime. For that reason, the precision requirement for LU decomposition remains the same. However, since the matrix to invert in (6) is not hermitian anymore, the LDLH algorithm cannot be applied for iterative detection. The same applies for the regularized QR decompositions (MGS and GR). Thus, the decomposition as in (12) has to be replaced by (10), which comes at the expense of higher precision requirements. As a result, the LU decomposition is the most efficient choice for fixed-point iterative detection.

	flp-19	fxp-18	fxp-20	fxp-22	fxp-24
f^A [MHz]	476	357	357	345	333
A^A_{nc} [kGE]	137	126	148	165	183
A^A_c [kGE]	-	146	166	205	214

TABLE III
POINTS OF MAXIMUM AREA EFFICIENCY

B. *Comparison of fixed-point and floating-point*

For open-loop detection at 64QAM, the energy efficiencies of the fixed-point and floating-point core almost coincide. However, when switching to lower order constellations, the floating-point variant gains a clear advantage, since operand masking can be applied more efficiently. This is due to the normalized nature of the floating-point number format, where the most significant bit (MSB) is always located at the same position within the mantissa representation. Note that this is not the case for fixed-point numbers, where the MSB can be located at any position within the data word. For iterative detection, the gap between floating-point and fixed-point efficiency widens due to the increased scaling effort in fixed-point arithmetic. Regarding operand masking, the same behavior shows as for open-loop detection.

In summary, the fixed-point and floating-point implementation deliver similar energy efficiency for open-loop detection while the floating-point variant has an advantage for iterative detection. At the same time, the floating-point core delivers approximately twice the throughput of the fixed-point variant. As a result and in contrast to common belief, the floating-point variant can be clearly identified as the preferable choice.

		4QAM		16QAM		64QAM	
	W_c [Bit]	E_{avg} [nJ]	τ [%]	E_{avg} [nJ]	τ [%]	E_{avg} [nJ]	τ [%]
Open-loop							
fxp-MGS (reg.)	20	29.0	11	30.3	6.7	31.6	2.6
fxp-GR (reg.)	16	50.1	9.3	52.9	14	55.3	0.0
fxp-LDLH	24	37.4	5.6	38.7	4.3	39.6	0.0
fxp-LU	24	36.7	6.6	38.1	2.2	39.3	0.0
flp-DnQ	19	23.5	28	30.4	7.3	31.8	3.0
Iterative							
fxp-MGS	24	39.9	7.8	41.1	5.1	42.5	1.7
fxp-GR	20	52.4	9.5	55.8	3.6	57.9	0.0
fxp-LU	24	35.7	5.1	36.9	1.9	37.6	0.0
flp-DnQ	19	26.7	23	29.7	15	32.8	5.7

TABLE IV
ENERGY EFFICIENCY OF 4×4 LINEAR MIMO DETECTION ALGORITHMS ON FIXED- AND FLOATING-POINT PROCESSOR CORE

The advantages of the floating-point core actually extend to aspects of programmability. Firstly, the programming effort of a numerically stabilized fixed-point application is significantly higher than for a floating-point application where stabilization is ensured by the arithmetic units themselves. On top of that, a significant part of the exploration effort has to be spend to figure out e.g. the position of the decimal point in the fixed-point application, which varies throughout different sections within one algorithm.

Despite this finding, fixed-point arithmetic should not be discarded for linear MIMO detection. Even though floating-point is superior in overall energy efficiency, fixed-point can still exceed in certain subtasks of linear detection. It is to be expected, that e.g. the calculation of $\mathbf{H}^H\mathbf{H}$ in (6) or $\mathbf{H}^H\mathbf{y}$ in the actual equalizing has a small dynamic range and will consume less energy when executed in fixed-point arithmetic. This aspect shall be further analyzed by looking at the calculation of different parts of (6).

C. Energy breakdown & channel condition

In the following, we look at the energy consumption of calculating

$$\mathbf{G} = \left(\mathbf{H}^H\mathbf{H}\boldsymbol{\Lambda} + N_0\mathbf{I}\right)$$

and the subsequent inversion delivering $\mathbf{G}^{-1}$ for both architectures. Input data is acquired from the first iteration with feedback information of a 4×4 MIMO transmission using a 16QAM Gray-coded constellation and convolutional channel coding at rate $r = 1/2$. Generator polynomials are chosen as $g_0 = (133)_8$ and $g_1 = (171)_8$ in line with the IEEE 802.11n standard. Channel decoding is performed by a soft-in soft-out BCJR algorithm [12]. The channel is modeled by an i.i.d. Rayleigh fading process superimposed by additive white Gaussian noise (AWGN) with a power delay profile as described by the TGn-C model [13]. Energy consumptions are shown along with the coded bit-error-rate (BER) in Fig. 5 as a function of the signal-to-noise ratio (SNR) per receive antenna.

Several conclusions can be drawn from this analysis. Firstly, we notice that the energy advantage of the floating-point architecture originates from the more efficient matrix inversion as opposed to the purely multiplicative part, which actually consumes more energy on the floating-point core. Secondly, we see that energy efficiency actually depends on the channel conditions. For both number formats, the inversion energy decreases for better channel conditions at higher SNRs. The exact slope of both inversion energies is notably different, though. With decreasing variance $\boldsymbol{\Lambda}$, matrix $\mathbf{W}^H$ in (6) converges to a scaled identity matrix. For higher SNRs, the fixed-point prescaling has to perform increasingly wide left shifts, meaning that the number of significant bits decreases, resulting in a diminishing switching activity and energy consumption. Floating-point arithmetic on the other hand calculates $\mathbf{W}^H$ at full precision, as long as the occurring numbers are still within the exponent's range. Hence there is less dependency of the energy efficiency on the channel conditions. As a result, the fixed-point implementation becomes more energy efficient than the floating-point variant for the given scenario at an SNR greater than approximately 21.5 dB.

Table V gives an overview of the energy distribution among different parts of both cores when executing iterative MIMO detection for 16QAM at 19 dB of SNR. While both of them spend most energy within the ALU, the higher effort for floating-point additions results in a higher fraction of energy in the adder tree. Also, the longer ALU and pipeline result in

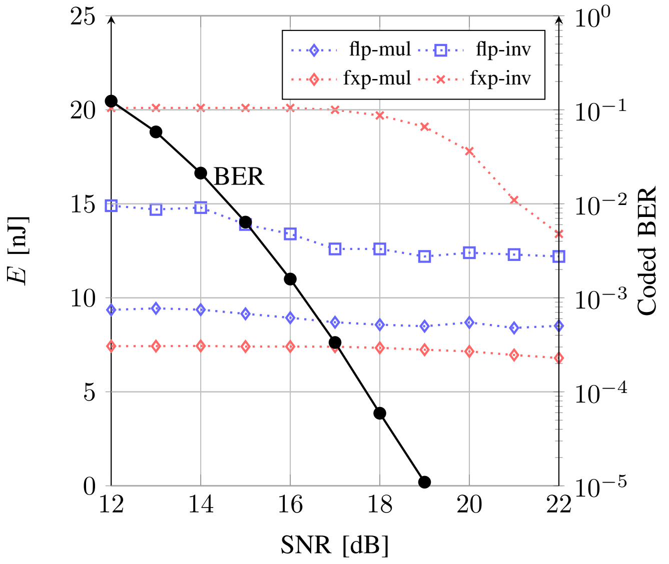

Fig. 5. Energy-SNR chart for fixed-point and floating-point core

	Floating-point	**Fixed-point**
Multiplier/Reciprocal	51.0	58.2
Adder tree	14.3	0.5
Register file	7.2	3.0
Pipeline register	14.1	10.2
Misc.	13.4	28.2

TABLE V
ENERGY BREAKDOWN (IN%) AMONG CORE SEGMENTS

a higher energy consumption within the pipeline registers of the floating-point core. Since the absolute amount of energy consumed by the ALU is lower for the fixed-point core, the percentage of energy consumed by miscellaneous operations (e.g. memory access) is notably higher.

VI. CONCLUSION

In this paper, we presented a fixed-point and a floating-point variant of a single-instruction multiple-data (SIMD) processor core tailored to complex-valued vector arithmetic in general and linear MIMO detection in particular and performed an in-depth analysis with respect to the achievable throughput and energy efficiency. To this end, numerical precision requirements for a wide set of matrix inversion algorithms were first investigated and both cores were synthesized for the corresponding word width. Based on the synthesized models, the most efficient inversion algorithms were selected and then used to compare the energy efficiency of MIMO detection for both number formats. In a further analysis, the detection task was subdivided into sections with high and low dynamic range to illustrate how each core performs under different conditions.

On an architectural level, we see that the fixed-point variant requires a more complex instruction set for numerical stabilization, e.g. for prescaling. Since mantissa scaling is inherent to the floating-point number format, the floating-point version does not require such features and the instruction-set was designed to be more generic. Additionally, both cores allow operand masking to reduce the number of significant bits in the fixed-point words and in the floating-point mantissa respectively. We observed that due to the normalized nature of the floating-point mantissa, this feature can be used more efficiently in the floating-point core, thus increasing its efficiency advantage when operating on a less dense modulation scheme with reduced precision requirements.

To evaluate each core's performance in the scope of linear MIMO detection, the algorithm of choice for matrix inversion, which makes up a significant fraction of the detection complexity, had to be determined for both number-formats. Since the high dynamic range of floating-point arithmetic requires less numerical stabilization, a simple divide-and-conquer algorithm was chosen. The lower dynamic range of fixed-point arithmetic on the other hand requires higher stabilization effort e.g. by a factorization-based matrix inversion. Various choices were evaluated regarding their energy efficiency, where the modified Gram-Schmidt scheme was found to be most efficient for open-loop detection and the LU algorithm proved most efficient for iterative detection.

Generally, the floating-point variant was found to have a higher power consumption due to the increased complexity of floating-point arithmetic. However, the lower stabilization effort enables an approximately twofold advantage in throughput. Energy consumption is approximately the same for both number formats when performing open-loop detection. However, for iterative detection, the additional stabilization effort of the fixed-point variant renders it inferior to the floating-point counterpart. A further energy breakdown showed that this advantage is mainly caused by the more efficient matrix inversion. Furthermore, the energy consumption showed a different dependency on the channel condition for both number formats. Due to the high dynamic range of the floating-point format, the mantissa multipliers always operate at full precision, resulting in a small dependency of the energy consumption on the channel condition. For fixed-point arithmetic on the other hand, the number of significant bits reduces gracefully, resulting in a steeper decay of energy consumption for better channel conditions. Although we saw that the floating-point core generally delivers superior efficiency and throughput over the fixed-point variant, it should be noted that the fixed-point core delivers better energy efficiency for certain subtasks with little dynamic range, e.g. the multiplicative part of linear detection. This finding motivates a hybrid architecture that deactivates the normalization logic within the floating-point units for those subtasks with a narrow dynamic range.

REFERENCES

[1] IEEE Standard for Information Technology–Telecommunications and information exchange between systems–Local and metropolitan area networks–Specific requirements Part 11, Amendment 5. *IEEE Std 802.11n-2009*, pages 1–502, Oct. 2009.

[2] J. Zyren. Overview of the 3GPP Long Term Evolution Physical Layer, Jul. 2007.

[3] D. Guenther, R. Leupers, and G. Ascheid. Mapping of MIMO Receiver Algorithms onto Application-Specific Multi-Core Platforms. In *The Tenth International Symposium on Wireless Communication Systems*, 2013.

[4] H. Blume, H. Hubert, H.T. Feldkamper, and T.G. Noll. Model-based exploration of the design space for heterogeneous systems on chip. In *Application-Specific Systems, Architectures and Processors, 2002. Proceedings. The IEEE International Conference on*, pages 29–40, 2002.

[5] David Tse and Pramod Viswanath. *Fundamentals of Wireless Communications*. Cambridge University Press, 2004.

[6] Xiaodong Wang and H.V. Poor. Iterative (turbo) soft interference cancellation and decoding for coded CDMA. *Communications, IEEE Transactions on*, 47(7):1046–1061, Jul. 1999.

[7] C. Studer, S. Fateh, and D. Seethaler. ASIC Implementation of Soft-Input Soft-Output MIMO Detection Using MMSE Parallel Interference Cancellation. *Solid-State Circuits, IEEE Journal of*, 46(7):1754–1765, Jul. 2011.

[8] Gene H. Golub and Charles F. Van Loan. *Matrix computations (3rd ed.)*. Johns Hopkins University Press, Baltimore, MD, USA, 1996.

[9] S. Eberli, D. Cescato, and Wolfgang Fichtner. Divide-and-Conquer Matrix Inversion for Linear MMSE Detection in SDR MIMO Receivers. In *NORCHIP, 2008.*, pages 162–167, nov 2008.

[10] Peter Jan Luethi. *VLSI circuits for MIMO preprocessing*. ETH, 2009.

[11] P. Luethi, C. Studer, S. Duetsch, E. Zgraggen, H. Kaeslin, N. Felber, and W. Fichtner. Gram-Schmidt-based QR decomposition for MIMO detection: VLSI implementation and comparison. In *Circuits and Systems, 2008. APCCAS 2008. IEEE Asia Pacific Conference on*, pages 830 –833, Dec. 2008.

[12] D.J.C. MacKay. *Information Theory, Inference and Learning Algorithms*. Cambridge University Press, 2003.

[13] E. Perahia and R. Stacey. *Next Generation Wireless LANs: Throughput, Robustness, and Reliability in 802.11n*. Cambridge University Press, Cambridge, UK, 2010.