Efficiency enablers of lightweight SDR for MIMO baseband processing

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

***Abstract*—The flexibility and programmability of an application-specific instruction-set processor (ASIP) come at the expense of reduced area and energy efficiency compared to application-specific integrated circuit (ASIC) solutions. Nevertheless, ASIPs are desirable for versatile application domains like wireless communications and software defined radio (SDR). Typically, ASIP designers reduce the ASIC-ASIP efficiency gap by increasingly complex architectures with decreasing flexibility and usability. This paper takes the opposite approach and presents concepts for a highly efficient, lightweight SDR ASIP. Efficiency enablers include simple but effective measures like a carefully chosen instruction set, optimized data access techniques for efficient utilization of functional units, and the use of flexible floating-point arithmetic with runtime-adaptive numerical precision. We present a conceptual processor core to show the impact of these measures and discuss its potential as well as limitations compared to tailored ASIC solutions. For demonstration, we choose the field of linear MIMO detection. We present synthesis results for several design versions in 90 nm CMOS technology and the corresponding energy benchmarks. Also, we show post-layout results for a selected design to demonstrate the feasibility of our concept.**

***Index Terms*—SDR, ASIP, floating-point, adaptive precision, area and energy efficiency**

I. Introduction

The variety of wireless mobile communication standards like IEEE 802.11a/b/g/n [1], GSM, UMTS, LTE, and LTE Advanced [2] has motivated the use of programmable platforms for physical layer processing. Instead of integrating one application-specific integrated circuit (ASIC) for each communication standard, software defined radios (SDRs) implement the different standards in software. A programmable platform increases flexibility and decreases time to market, but has drawbacks in terms of efficiency, manifested as an increase in silicon area and energy consumption.

The simple and repetitive control flow of an ASIC can be represented by a finite state machine (FSM) which controls the combinational logic elements in the data path. This hard-wired control is more energy efficient than reading instruction words from a program memory and decoding them. Moreover, ASIC designers can reduce energy consumption by adapting the wordwidth of each arithmetic unit in the data path to the individual numerical requirements at design time, while processor cores for baseband processing are commonly limited to one or very few integer or fixed-point formats. To mitigate the ASIC-ASIP efficiency gap, ASIP designers have developed increasingly complex architectures like coarse grained reconfigurable arrays (CGRAs) controlled by an ASIP [3] or stream based architectures [4]. While improving efficiency, these architectures suffer from a loss of flexibility and programmability. In this paper, we take the opposite design approach and present a fully and easily programmable, flexible lightweight SDR ASIP and its efficiency enabling concepts. We show that a well designed lightweight ASIP surpasses other less flexible programmable solutions from the open literature and achieves a competitive efficiency when compared to tailored ASICs.

An efficient ASIP needs a suitable instruction set, versatile enough to support a multitude of use cases but also application-specific enough to boost the processor's efficiency into the range of comparable ASICs. The vectorial nature of multiple-input multiple-output (MIMO) baseband processing motivates a single instruction multiple data (SIMD) instruction set with native support for complex-valued arithmetic. To support e.g. multiple antenna configurations, the instruction set has to handle a set of matrix and vector dimensions efficiently. This calls for tailored permutation units to map the desired functionality onto the existing data path. Also, to ensure high utilization of the available functional units, a specialized bypassing unit that can retrieve computational results from different points within the pipelined arithmetic logic unit (ALU) is needed. The limited dynamic range of fixed-point number formats requires additional effort for numerical stabilization (e.g. by scaling or matrix factorization) [5], which can be avoided by the use of floating-point arithmetic. Despite the increased energy consumption per operation, the higher dynamic range enables the use of algorithms with reduced runtime [6], which puts this drawback into perspective.

MIMO baseband processing algorithms show diverse requirements in numerical precision depending on the use case (e.g. antenna setup). Moreover, some of these algorithms can be decomposed into distinct sections with different precision requirements. This inspired the concept of numerically aware processing (NAP), which adapts the numerical precision of the data path at runtime on a bit-granular level to reduce switching activity and hence energy consumption. The idea of NAP is related to the concept of approximate computing (AC) [7] which assumes that a small degradation of processing accuracy is tolerable e.g. due to perceptual limitations of humans with regard to multimedia content. Our research shows that the same concept applies to MIMO baseband processing.

In this paper, we present the *napCore*, a fully programmable floating-point processor core that implements aforementioned efficiency enabling measures. The core is generally suitable for algorithms based on vector arithmetic. We choose linear MIMO detection as an exemplary application domain, due to its popularity and the availability of implementations for comparison. Similar results can be obtained for other vectorial algorithms (e.g. linear channel estimation and interpolation). We show that a well designed lightweight ASIP can compete with less flexible architectures and non-programmable ASICs in terms of efficiency. While not each enabling measure on its own may be new, it is our main contribution to show that the right mix generates an architecture which proves that efficiency does not require complexity and does not contradict an architecture which is easy to use. In Section II, we give a general overview of the *napCore* architecture and its

efficiency-enabling features. Section III shows an architectural exploration by comparing energy efficiency and area efficiency of topographical gate-level models for varying operand widths. In Section IV, we present linear MIMO detection as a case study. After elaborating on the numerical precision requirements of the software implementation, we present a layout of our processor core for these requirements and discuss energy efficiency as well as area efficiency of the implementation.

II. Architecture overview

The *napCore* is a fully programmable SIMD processor core designed for vector arithmetic. This section discusses the enabling architectural features for area and energy efficiency while abstracting from any specific application. The data path of the *napCore* provides native support for complex-valued arithmetic. Its runtime-adaptive floating-point format can be changed within one clock cycle by means of mantissa masking (see Section II-E). The design contains separate memories for program data and vector data. The latter is a two-port memory with one read port and one write port. It stores data words of the same width as the operand vectors, so one vector can be read/written per memory operation. The architecture contains two register files. The scalar register file has two read ports and one write port. It is mainly used for scalar operations or vector arithmetic including a scalar operand. The vector register file has three read ports and one write port. It is internally realized as P scalar register banks, where P is the parallelism degree of the SIMD architecture. This banked design enables access to individual scalars without the need to read/rewrite the remaining vector elements. For the following case study in the field of MIMO baseband processing, we chose parallelism degree $P = 4$, so that one vector register can accommodate e.g. one matrix of dimension 2×2. This choice enables to use of efficient divide-and-conquer matrix operations for higher dimensions (see Section IV-B).

A. Pipeline overview

Figure 1 shows the pipeline structure of the SIMD core. An instruction word is requested from the program memory (*PMEM*) in the pre-fetch stage (*PFE*) and received one cycle later in the fetch stage (*FE*). It is then interpreted in the decode stage (*DC*), which configures all further stages. Operands are loaded and preprocessed by the *PrepOp-DC* unit, which also performs operand bypassing to resolve data hazards. The following four arithmetic stages (*EX1*, *EX2*, *RED1*, *RED2*) are designed to match the processing scheme of standard vector arithmetic operations, which is a composition of multiplications and subsequent additions. The additional complexity of floating-point additions over fixed-point additions motivates to split a complex-valued multiplication across stages *EX1* and *EX2*, where *EX1* executes the real-valued multiplications, while *EX2* accumulates the real-valued products to form the complex-valued result. Newton-Raphson units for scalar inversion are also located in *EX1*. In the reduction stages *RED1* and *RED2*, the result of the element-wise multiplication can be further processed by means of additions, which can be configured e.g. to form an adder tree. Also, one additional vector operand can be read from the vector register file by the *PrepOp-EX2* unit in *EX2* to serve as input to *RED1*, e.g. for multiply-accumulate operations. Results are written back to the vector memory (*VMEM*) or the scalar/vector register files after processing is completed in the *RED2* stage.

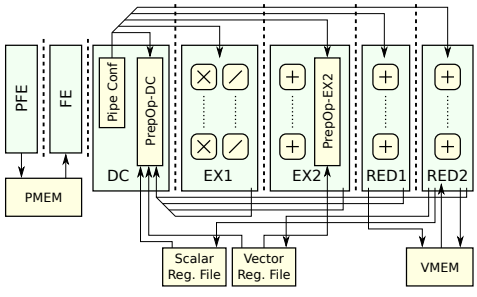

Fig. 1: Overview of SIMD processor core architecture

B. Operand acquisition

For programmable architectures with inherent parallelism like SIMD or very long instruction word (VLIW) processors, the potential for data-level parallelism is defined by the parallelism of the data path, given there is an efficient operand acquisition mechanism. Even for regular vector arithmetic operations, this is a challenging task. Consider the previously described SIMD architecture with a scalar and a vector register file. Depending on the instruction, very different data access patterns have to be realized, which leads to the complex operand acquisition architecture depicted in Fig. 2 for the first operand. Widths of the data path are given as multiples of complex-valued scalars.

The vector and scalar register files *vREG* and *sREG* are accessed depending on the type of the operand. Bypassing units *vBP* and *sBP* attempt to obtain the demanded operand from the pipeline where it is potentially present as the result from a previous operation (see Section II-D) and signal the successful acquisition (*is_bp*) to the subsequent multiplexer. In the next step, the required scalar elements are sent to the four mantissa masking units (marked by &). The switch *s3* configures if the operand to load is scalar or vectorial and accordingly activates the first or all masking units. A scalar operand, however, can come from a scalar register or one element of a vector register as configured by *s2*. If the scalar comes from the vector register, the right element is selected by *s1*. The masked operands are forwarded to the pipeline register between stages *DC* and *EX1*. In case the same scalar element is to be forwarded to all elements of the operand, as e.g. required for a scalar-vector multiplication, this can be triggered by *s4*. Otherwise the masking results are forwarded element by element. When fetching the operand in the *EX1*

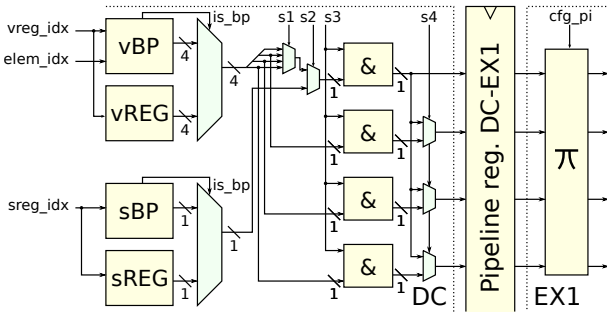

Fig. 2: Schematic of data acquisition for first operand

stage, there is an additional permutation unit marked as π (see Section II-C). This flexible operand acquisition paired with the P-wise partitioned vector register file allows versatile data access schemes, e.g. obtaining scalars from the scalar register file or from elements of the vector register file, the use of vector elements in scalar operations, and using scalar elements (from *vREG* as well as *sREG*) in vectorial operations.

C. Permutation network

Figure 3 shows a schematic overview of the two permutation networks in front of the multipliers in *EX1*. Apart from straight pass-through, the networks support patterns especially for 2×2 vector arithmetic operations like matrix inversion, determinant calculation, or matrix-matrix multiplication. Since the first vector typically holds the left-hand value of a multiplication and 2×2 matrices are stored row-wise in the vector registers, the left and right pair of multiplexers are wired to select one of the two matrix rows via *hilo1* and *hilo2*. Furthermore, the crossbar *cb1* allows to repeat the same scalar element twice at the output. The path for the second vector operand contains the same set of multiplexers. Since the right-hand value of a matrix multiplication is typically accessed column-wise while the matrix is stored row-wise, the input can be transposed via crossbar *cb2*. Crossbar *cb3* is used to realize further permutations, e.g. for 2×2 matrix inversion.

D. Operand bypassing

The bypassing unit attempts to fetch an operand from the pipeline as a result of a previous instruction instead of waiting until it becomes available in the register file after passing through the pipeline. Since the ALU of the *napCore* architecture spans across several pipeline stages, the decoder has to know for each instruction at which stage of the pipeline its computation is finished. When executing a component-wise complex multiplication of two vectors for example, the result is valid after the summation of the partial products in the *EX2* stage and can be bypassed from there. To inform the decoder about this context, each instruction injects an index into the pipeline at the decoding stage, indicating after how many arithmetic stages its result is valid. The bypassing logic in the decode stage uses this information to decide whether or not an operand is bypassed. Since an additional vector operand can be loaded as input to the *RED1* stage, one further operand preprocessing unit *PrepOp-EX2* is located in the *EX2* stage. The major difference between *PrepOp-DC* and *PrepOp-EX2* is that there is no bypassing in the latter, and it obtains just one vector from the vector register file.

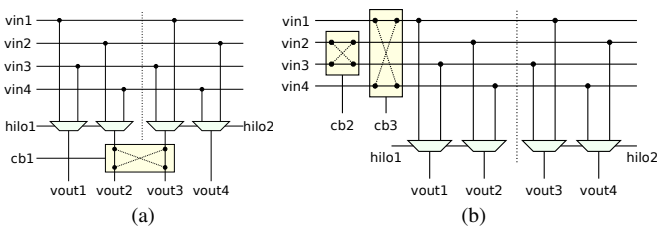

Fig. 3: Permutation units for vector operands one and two

E. Numerically aware processing

Many algorithms can be decomposed into distinct sections of different requirements for numerical precision, as we will demonstrate in Section IV-B for the exemplary case of two linear MIMO detection algorithms. For a floating-point number

$$a = sgn \cdot m \cdot 2^e \tag{1}$$

with sign $sgn \in \{+1, -1\}$, mantissa m, and exponent e, we can exploit the normalized nature of the floating-point mantissa, which guarantees $1 \leq m < 2$. We know that masking a certain number of least significant bits (LSBs) of the mantissa, in the following referred to as *mantissa masking*, will always leave the same number of most significant bits (MSBs). Note that this is not the case for fixed-point data formats, where the MSB may be located at any bit position within a data word. Accordingly, mantissa masking is more suitable for floating-point number formats. The principle is illustrated in Fig. 4, where a variable bitmask is applied to the last four LSBs of an exemplary floating-point number format. The width of the bitmask has to be chosen according to the variation of precision requirements in the target application domain. In our processor core, we place a masking unit as in Fig. 4 at the end of operand loading in *PrepOp-DC* as well as after every arithmetic component within the 4-stage ALU. The bitmask can be adapted at runtime by a configuration instruction in the program code.

F. Floating-point Newton-Raphson iterator

One common task in vector arithmetic is vector norming, which requires the calculation of a scalar inverse. This operation is one of the few examples where the floating-point arithmetic unit is less complex than its fixed-point counterpart, as will be illustrated in the following. Since root finding problems like scalar inversions are computationally complex, they are typically approximated by the Newton-Raphson algorithm [8], which finds the nulls of a function f by iteratively calculating

$$y_{n+1} = y_n - \frac{f(y_n)}{f'(y_n)} \quad n = 0, 1, 2, .. \tag{2}$$

given an initial choice of y_0, where f' is the derivative of function f. To find $y = 1/x$, the nulls of the function $f(y) = 1/y - x$ have to be determined. In that case, the iteration according to (2) is given by

$$y_{n+1} = 2y_n - y_n^2 x, \tag{3}$$

where our processor core calculates one iteration per cycle. The choice of a suitable initial y_0 close to the converging point of the iteration is essential for fast convergence. For a floating-point number as in (1), the scalar inverse is calculated as

$$\frac{1}{a} = sgn \cdot \frac{1}{m} \cdot 2^{-e}. \tag{4}$$

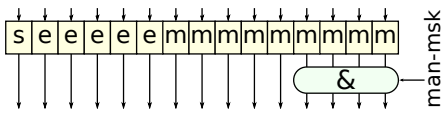

Fig. 4: Mantissa masking

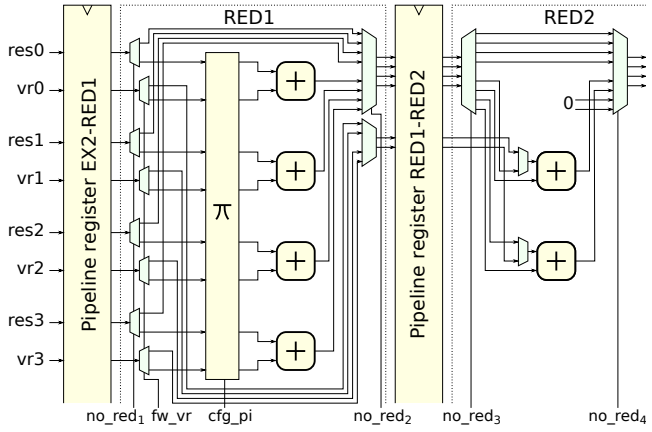

Fig. 5: Reduction stages *RED1* and *RED2*

Since mantissa m lies in the range of $1 \leq m < 2$, the inverse mantissa is also limited in range $(0.5 < 1/m \leq 1)$. Thus, the selection of y_0 is less complex than for a fixed-point implementation, where the range of the input operand has to be considered explicitly. This allows the implementation of a simple two-choice selection mechanism, dividing the solution space into two parts of equal size. Four subsequent Newton-Raphson iterations are sufficient for the baseband applications described in Section IV-A.

G. Configurable reduction stages

To support a versatile instruction set, e.g. for efficient processing of vectorial data of different dimensions, the reduction stages *RED1* and *RED2* are designed to fit the requirements of a wide range of vector arithmetic operations. The maximum number of required complex adders in *RED1* corresponds to SIMD parallelism degree P, which is needed, if a multiply-accumulate operation with P-dimensional vector operands is executed. Note that for an inner product, an adder tree of depth $\mathrm{ld}\,(P)$ is sufficient, which requires $P/2$ adders in *RED1* (if P is a power of 2). In *RED2*, $P/4$ adders are sufficient for an inner product, but for our use case of $P = 4$, we chose to place an additional adder, which is used for some specialized instructions for $\sqrt{P} \times \sqrt{P}$ vector arithmetic (the dimension for which one square matrix fits into one vector register). Figure 5 shows parts of the reduction stages *RED1* and *RED2* for $P = 4$. For reasons of simplicity, not all multiplexing and demultiplexing control signals are included. As previously discussed, there are four and two complex adders in *RED1* and *RED2*, respectively. To support various vector arithmetic operations, the additions in stages *RED1* and *RED2* require flexible interconnects. The input to *RED1* can come from the result of previous pipeline stages (*res0..res3*) or from the vector register file (*vr0..vr3*), whose content can also be forwarded to *RED2* (*fw_vr*). If no additions are required for the current instruction, the result from *EX2* can also be simply forwarded through the pipeline by setting $no\_red_1..no\_red_4$. As will be shown in Section IV-B, support for $\sqrt{P} \times \sqrt{P}$ vector arithmetic enables efficient divide and conquer algorithms, e.g. for matrix inversion. The price to pay to support these operations is the permutation network marked as π in front of the adders in Fig. 5, configurable via *cfg_pi*.

III. Synthesis Results & Energy Benchmark

This section discusses synthesis results and identifies design points optimizing throughput, area efficiency, or energy efficiency respectively. Based on this analysis, the potential for energy savings using mantissa masking is evaluated.

A. Design space exploration

In Fig. 6, we present synthesis results of the *napCore* architecture for different floating-point formats. Here, *s* stands for the sign bit, *m* represents the number of mantissa bits, not including the redundant leading MSB (hidden bit), and *e* stands for the number of exponent bits. The design is synthesized for a 90 nm standard performance CMOS technology under typical conditions with a core voltage of 1 V. On the Y-axis, the figure shows the silicon complexity A as number of required kilo gate equivalents (kGE) without memory as a function of the clock period T_c on the X-axis. The figure allows the identification of the design points with maximum architectural throughput as well as the minimum area-timing

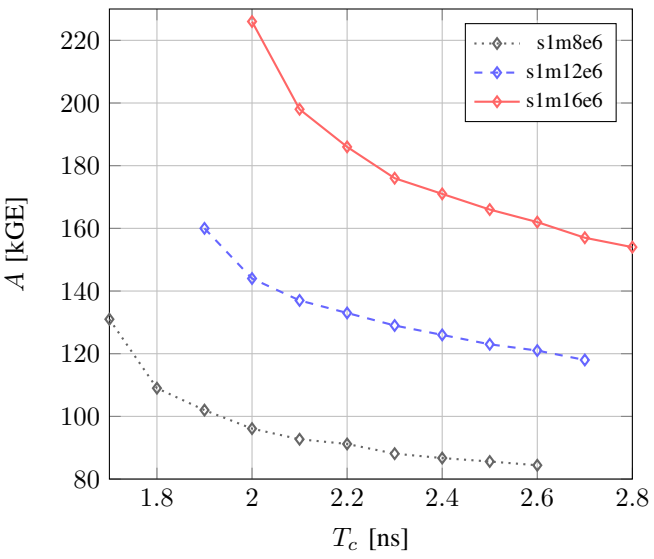

Fig. 6: Synthesis results: area efficiency

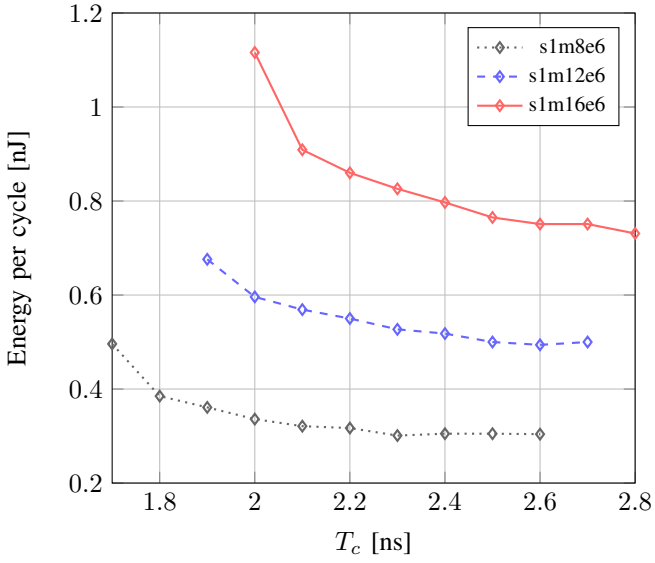

Fig. 7: Synthesis results: energy efficiency

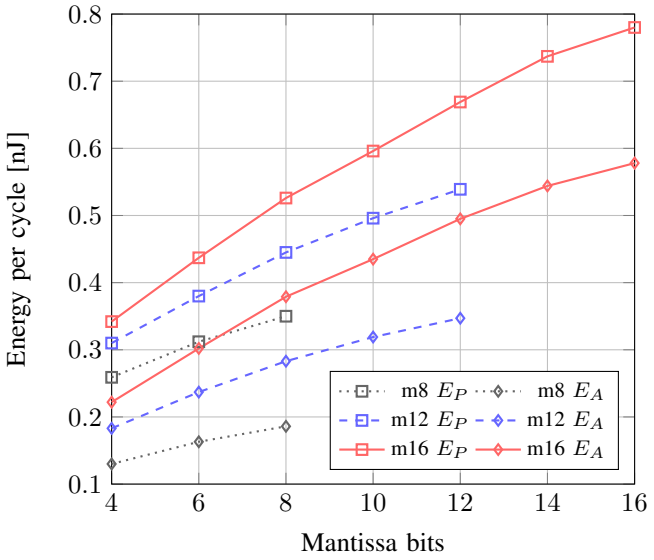

Fig. 8: Energy consumption per cycle under full load benchmark for s1m8e6, s1m12e6 and s1m16e6

(AT) products, which represent the design points for maximum area efficiency.

Figure 7 shows the results of an energy benchmark for the same clock frequencies and number formats as presented in Fig. 6. Results are derived based on averaged power analysis of topographical gate-level models of the processor architecture. The benchmark program consists of a series of matrix-vector multiplications implemented by inner products, thus, it occupies all ALU stages of the pipeline simultaneously and is a good indicator for the energy consumption under full load. The energy E given on the Y-axis shows the average energy consumption per clock cycle. Naturally, a relaxed timing constraint during synthesis allows for a more energy efficient hardware implementation, but the decrease of $E(T_c)$ shows saturating behavior, so at a certain point, the marginal decrease in energy consumption does not justify lowering the clock frequency. As a result, this plot allows the identification of the design points for energy efficiency in the saturation area of $E(T_c)$. The results from Fig. 6 and Fig. 7 are summarized in Table I, giving the clock frequency at the design points for maximum throughput f_c^T, maximum area efficiency f_c^A, and energy efficiency f_c^E as well as the corresponding silicon complexities A^T, A^A and A^E.

Table II gives a more detailed overview of the power breakdown of the *napCore* architecture for number formats *s1m8e6*, *s1m12e6* and *s1m16e6*. Here, as well as for the remainder of this paper, all configurations are synthesized

	s1m8e6	s1m12e6	s1m16e6
f_c^T	588	526	476
f_c^A	500	476	435
f_c^E	435	400	385
A^T	131	160	198
A^A	96.1	137	176
A^E	88.1	123	162

TABLE I: Clock frequency f_c [MHz] and area A [kGE] for different design points

at the design point for energy efficiency. The table shows the power consumption of the arithmetic stages, the program memory and the register file as well as further miscellaneous parts like control and pipeline registers. For all configurations, the major share of the power is consumed by the 16 multipliers in the *EX1* stage, and their share increases when raising the number of mantissa bits from eight (33.9%) to 16 (51.2%). The second largest share comes from the floating-point adders in the remaining ALU stages. A fraction of around 62 to 74% of the total energy is consumed within the ALU. The remains is distributed among control, memory accesses, register file accesses, and switching activity within pipeline registers.

B. Energy benchmark for mantissa masking

To assess the potential energy savings achievable by means of adaptive mantissa precision, we execute the same benchmark as presented in Section III-A for core configurations *s1m8e6*, *s1m12e6* and *s1m16e6*. We reduce the mantissa width bit by bit in software at runtime and observe how the energy consumption changes. The results are shown in Fig. 8. The first triplet of lines (suffix E_P) shows the energy consumption of the entire processor core, while the second triplet (suffix E_A) solely shows the energy consumed within the arithmetic logic stages EX1, EX2, RED1, RED2.

Figure 8 outlines the potential as well as limitations of mantissa masking. On the one hand, it demonstrates that the runtime-adaptive mantissa format enables energy savings which show nearly linear behavior with respect to the chosen wordwidth. On the other hand, we see that there is still an overhead of masking down the mantissa format as opposed to using arithmetic units which natively support the required precision. Observe e.g. the difference in energy consumption per cycle between core configurations *s1m16e6* and *s1m12e6* for 12 mantissa bits. For *s1m16e6*, mantissa masking decreases energy consumption from 0.780 nJ to 0.669 nJ, while in native *s1m12e6* arithmetic, the operation only consumes 0.539 nJ.

IV. Case Study: Linear MIMO Detection

In the following case study, we apply the concept of NAP to linear MIMO detection. First, two linear MIMO detection algorithms are presented. Then, their software implementation on the *napCore* is discussed, including an evaluation of the required numerical precision. After that, a layouted version of the processor core for the maximum required precision is presented and used to evaluate the efficiency of the implementations of the previously introduced algorithms.

	s1m8e6		s1m12e6		s1m16e6	
	mW	%	mW	%	mW	%
EX1	44.4	33.9	90.4	45.2	148.0	51.2
EX2	15.8	12.1	20.3	10.1	27.4	9.5
RED1	9.6	7.3	13.2	6.6	18.4	6.4
RED2	11.2	8.5	14.9	7.4	20.0	6.9
PMEM	9.6	7.3	8.9	4.5	8.6	2.9
REG	11.5	8.8	13.5	6.8	17.3	6.0
MISC	28.9	22.1	38.8	19.4	49.3	17.1
Σ	131		200		289	

TABLE II: Power breakdown under full load benchmark

A. Algorithms

Multicarrier transmission schemes like orthogonal frequency-division multiplexing (OFDM) divide the available bandwidth into approximately frequency-flat subcarriers. This allows the description of the transmission of each subcarrier separately. For a system with N_t transmitter antennas and N_r receiver antennas, these frequency-flat transmissions are given by

$$\mathbf{y} = \mathbf{H}\mathbf{x} + \mathbf{n}, \tag{5}$$

where $\mathbf{x}$ is the transmitted symbol vector of dimension N_t, $\mathbf{y}$ is the received symbol vector of dimension N_r, $\mathbf{H}$ is the channel matrix of dimension $N_r \times N_t$ of the current subcarrier, and $\mathbf{n}$ is the noise vector of dimension N_r.

Figure 9 shows the basic structure of a MIMO OFDM receiver using linear MIMO detection. After translating the received signal from analog to digital domain (A/D), RX OFDM processing transfers the incoming time domain data to frequency domain. After that, the non-guard subcarriers are forwarded to the subsequent receiver chain. MIMO preprocessing is a preparative step to enable actual MIMO detection. First, an estimate of the channel matrices is derived for each subcarrier. Then, an equalizer matrix (EQM) is calculated and used by the spatial equalizer to estimate the originally transmitted symbol vector. The soft symbol demapper transfers this vector into a series of log-likelihood ratios (LLRs). These soft-bits are then handed to the outer modem for forward error correction (FEC). In the next detector-decoder iteration, the result of FEC is fed back into the receiver chain.

1) Open-loop linear MMSE detection: Linear MMSE equalization derives an estimate $\hat{\mathbf{x}}$ of the originally transmitted symbol vector $\mathbf{x}$. The equalizer matrix is chosen to minimize the expected mean square error between $\mathbf{x}$ and $\hat{\mathbf{x}}$. For a flat fading channel superimposed by additive white Gaussian noise (AWGN) of spectral density N_0, the resulting equalizer matrix is derived as

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

where $\hat{\mathbf{H}}$ is the estimated channel matrix and $\mathbf{I}_{N_t}$ is an identity matrix of dimension $N_t \times N_t$. After multiplying the equalizer matrix by the received symbol vector, the estimate $\hat{\mathbf{x}}$ is transferred back to a bitwise representation. For that purpose, an LLR is derived for each bit i of each antenna element k of the symbol vector according to

$$L(b_{k,i}) \approx \rho_k \left(\min_{s \in A_i^0} |\hat{x}_k - s|^2 - \min_{s \in A_i^1} |\hat{x}_k - s|^2 \right), \tag{7}$$

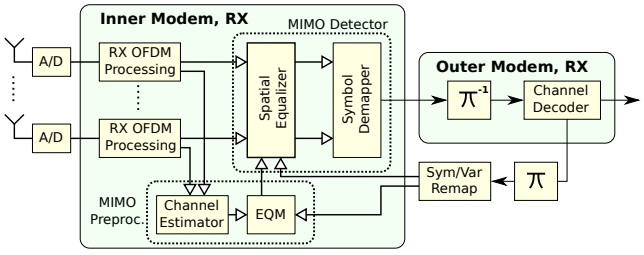

Fig. 9: MIMO OFDM receiver structure

with A_i^0 denoting 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. Finally, the signal-to-interference-plus-noise ratio (SINR) ρ_k of the k-th antenna stream is calculated according to

$$\rho_k \approx \frac{1}{N_0 \left[\left(\hat{\mathbf{H}}^H\hat{\mathbf{H}} + N_0\mathbf{I}_{N_t}\right)^{-1}\right]_{k,k}} - 1. \tag{8}$$

As an extension to our work presented in [6], we decompose the equalization and SINR calculation into three classes of different numerical precision requirements. The first one, $\mathcal{P}_{ol}^{mul}$, is used in the multiplicative part of equalization, namely calculating

$$\mathbf{A}_{ol} = \left(\hat{\mathbf{H}}^H\hat{\mathbf{H}} + N_0\mathbf{I}_{N_t}\right) \tag{9}$$

and subsequently

$$\mathbf{y} = \mathbf{A}_{ol}^{-1}\hat{\mathbf{H}}^H\mathbf{y}.$$

The class $\mathcal{P}_{ol}^{inv}$ covers the calculation of $\mathbf{A}_{ol}^{-1}$ itself, while $\mathcal{P}_{ol}^{llr}$ includes deriving the SINR as in (8), which is later used for LLR calculation. The precision requirements of these classes will be presented at the end of this section.

2) Iterative linear MMSE detection and MMSE-PIC: Iterative linear MIMO equalization derives an estimate for the transmitted symbol vector based on the estimated channel state ($\hat{\mathbf{H}}$, N_0) and additional priori information from previous iterations, e.g. in the shape of posteriori LLRs. The concept is briefly introduced in the following. The interested reader is referred to [9] for more information on iterative linear MMSE detection and to [10] for further information on the reduced complexity parallel interference cancellation (MMSE-PIC) algorithm, as it is used in this work.

The algorithm presented in [10] requires to first derive the most likely transmitted symbol vector $\mathbf{s}$ based on posteriori LLRs. In [11], it is described how the computational complexity of this vector remapping can be reduced to a piecewise linear function. Assuming that $\mathbf{s}$ was actually transmitted, the transmitter symbol vector estimate is now used to cancel the inter-antenna interference it would have caused at the receiver. The resulting set of interference mitigated receiver symbol vectors is given by

$$\hat{\mathbf{y}}_k = \mathbf{y} - \sum_{j \neq k} \mathbf{h}_j s_j, \tag{10}$$

where $\mathbf{h}_j$ denotes the j-th column of the estimated channel matrix. The vectors $\hat{\mathbf{y}}_k$ with $k \in \{1..N_t\}$ are an approximation of what the receiver antennas would have received, if only the k-th transmitter antenna had been transmitting. Afterwards, MMSE filtering is applied to these vectors to derive an estimate of each transmitted symbol vector element. It was shown in [10] that the required MMSE filter vectors can be derived from a single matrix

$$\mathbf{W}^H = \left(\hat{\mathbf{H}}^H\hat{\mathbf{H}}\mathbf{\Lambda} + N_0\mathbf{I}\right)^{-1}\hat{\mathbf{H}}^H. \tag{11}$$

The diagonal matrix $\mathbf{\Lambda}$ of dimension $N_t \times N_t$ holds the element-wise symbol variances of $\mathbf{s}$. These variances can be calculated based on the LLRs of the previous iteration by

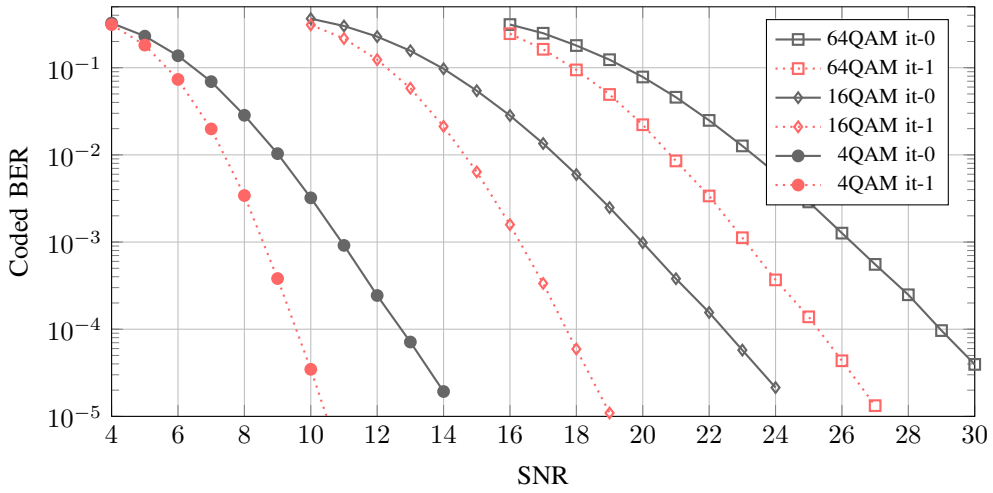

Fig. 10: Coded bit error rate for open-loop and iterative 4×4 MIMO detection using 4QAM, 16QAM and 64QAM constellations[1]

means of piecewise linear functions [11]. The actual MMSE filtering can be expressed based on the rows $\mathbf{w}_1^H$ .. $\mathbf{w}_{N_t}^H$ of $\mathbf{W}^H$ according to

$$\hat{\mathbf{x}}_k = \frac{\mathbf{w}_k^H \hat{\mathbf{y}}_k}{\mathbf{w}_k^H \mathbf{h}_k}. \tag{12}$$

The new LLRs for the i-th bit within the k-th element of vector $\hat{\mathbf{x}}$ are calculated as in (7) using the SINR given by

$$\rho_k = \frac{\mathbf{w}_k^H \mathbf{h}_k}{1 - \mathbf{\Lambda}_{kk} \mathbf{w}_k^H \mathbf{h}_k}. \tag{13}$$

Similar to Section IV-A1, we define three different classes of precision requirements for iterative MIMO detection. The class $\mathcal{P}_{it}^{mul}$ covers the calculation of

$$\mathbf{A}_{it} = \left(\hat{\mathbf{H}}^H \hat{\mathbf{H}} \mathbf{\Lambda} + N_0 \mathbf{I}_{N_t}\right) \tag{14}$$

as well as the interference mitigated vectors $\hat{\mathbf{y}}_k$ in (10). The class $\mathcal{P}_{it}^{inv}$ contains the actual matrix inversion of $\mathbf{A}_{it}$ and $\mathcal{P}_{it}^{llr}$ contains the calculation of SINRs, which are used later for LLR computation.

B. Software implementation

For a software implementation, the previously introduced algorithms have to be mapped onto our proposed processor architecture. A SIMD core is inherently suitable for vector arithmetic like the calculation of $\mathbf{A}_{ol}$ in (9) and $\mathbf{A}_{it}$ in (14). As discussed in Section II-G, our processor core supports 2×2 matrix-matrix operations, where each matrix is stored in a single vector register. These operations enable an efficient implementation of higher order matrix inversions, using a divide and conquer (DnQ) algorithm [4], which divides e.g. the inversion of a 4×4 matrix $\mathbf{A}$ into operations on 2×2 matrices $\mathbf{a}$, $\mathbf{b}$, $\mathbf{c}$ and $\mathbf{d}$.

$$\mathbf{A} = \begin{pmatrix} \mathbf{a} & \mathbf{b} \\ \mathbf{c} & \mathbf{d} \end{pmatrix} \tag{15}$$

Based on this representation, $\mathbf{A}^{-1}$ is given by

$$\mathbf{A}^{-1} = \begin{pmatrix} \mathbf{a}^{-1} + \mathbf{a}^{-1}\mathbf{bsca}^{-1} & -\mathbf{a}^{-1}\mathbf{bs} \\ -\mathbf{sca}^{-1} & \mathbf{s} \end{pmatrix} \tag{16}$$

with $\mathbf{s} = (\mathbf{d} - \mathbf{ca}^{-1}\mathbf{b})^{-1}$. The algorithm can be simplified for hermitian matrices as in (9) since $\mathbf{c} = \mathbf{b}^H$.

For the inversion of 2×2 matrix $\mathbf{a}$,

$$\mathbf{a}^{-1} = \frac{1}{\det \mathbf{a}} \begin{pmatrix} +a_{22} & -a_{12} \\ -a_{21} & +a_{11} \end{pmatrix}, \tag{17}$$

the determinant of $\mathbf{a}$ can be directly calculated due to the higher dynamic range of floating-point arithmetic. Thus, MIMO detection for multiple antenna setups (e.g. 2×2, 2×4, 4×4 and 4×8) can be efficiently implemented based on (16) and (17). Execution times of MIMO detection including equalizing and SINR calculation are listed in Table III for the aforementioned setups.

In the following, we analyze the numerical precision requirements of the previously introduced six classes, when using a DnQ matrix inversion algorithm. The analysis is based on extensive Monte Carlo simulations for a 4×4 MIMO setup with an i.i.d. Rayleigh slow fading channel superimposed by additive white Gaussian noise (AWGN). The power delay profile of the channel impulse response is modeled according to the TGn-C model [12]. The receiver setup is chosen as depicted in Fig. 9 with convolutional channel coding of rate $r = 1/2$, using generator polynomials $g_0 = (133)_8$ and $g_1 = (171)_8$. The code length is 3072, 6144 and 9216 bit for 4QAM, 16QAM and 64QAM[2]. Channel decoding is performed according to the BCJR algorithm [13]. In our analysis, we reduced the number format of each precision class until it caused a perceivable impact on the coded bit error rate (BER)

[1]it-0 and it-1 denote initial open-loop detection and first iterative detection
[2]Code length corresponds to eight OFDM symbols with 48 data subcarriers

Antenna setup	2 x 2	2 x 4	4 x 4	4 x 8
Open-loop	22	24.5	80	101
Iterative	32.5	35	112	137

TABLE III: Cycle count of linear MIMO detection

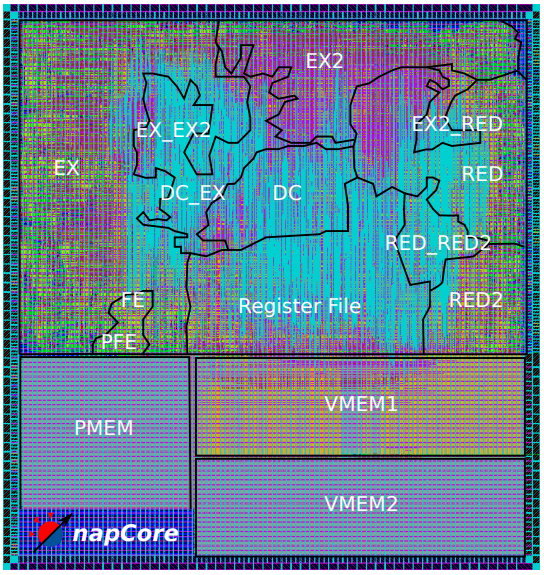

Fig. 11: Layout of *s1m12e6* configuration for energy efficiency

shown in Fig. 10. Table IV shows the required mantissa bits for all precision classes.

One sees an increase in precision requirements for more dense constellations, due to the narrower margin of error. The matrix inversion generally turns out to have a higher precision requirement than the multiplicative section because of the higher dynamic range, which can be mainly located within the determinant calculation. Only the SINR calculation has a constant precision requirement, since the channel decoder's algorithmic performance depends on the LLR precision and not on the used constellation. Also, it is interesting to observe the drop in precision requirements from the initial open-loop detection to the subsequent iterative detection. These findings can be exploited by the flexible characteristics of the *napCore* to reduce energy consumption.

C. Layout implementation

Table IV identifies the *s1m12e6* floating-point number format as the maximum precision requirement for the previously introduced linear detection algorithms. To prove the feasibility of our processor architecture, this section presents a post-layout model for that particular precision at the design point for energy efficiency. For increased accuracy, all following assessments of energy efficiency are based on that model. Figure 11 shows the physical view of the core, overlaid with the interpolated borders between the pipeline stages, pipeline registers (e.g. *DC_EX* marks the pipeline register between the *DC* stage and the *EX* stage) as well as program and vector memory. The layout achieves the target frequency of $400\,\mathrm{MHz}$. One clearly sees the high wiring effort on the top metal layer (in turquoise) to connect the vector register file to the *DC* stage, as well as the strategic placement of the *DC* stage itself in the middle of the design to facilitate vector bypassing from all computational stages back to *DC*. Another contribution to the dense wiring in the *DC* stage results from the flexible operand acquisition logic described in Section II-B. Also the register file is strategically located close to the *DC* stage for operand loading as well as to the *RED2* stage for operand writeback.

The design presented in Fig. 11 contains a vector memory for 512 vector words of length 152 bit, which is partitioned into two memory banks. The instructions are fetched from a dedicated program memory with space for 1024 words of 32 bit. Note that for the sake of flexibility and extendability, the program memory size is significantly higher than the number of instructions required for MIMO detection. Similarly, the aforementioned detection implementations operate on the register files exclusively and the vector memory is only included to support further potential applications with higher caching needs. The total design dimension (including memory and power rings) is $965\,\mu\mathrm{m}$ by $1020\,\mu\mathrm{m}$ resulting in a total area of $0.984\,\mathrm{mm}^2$. The upper part containing the pipeline and the register file measures $597\,\mu\mathrm{m}$ by $905\,\mu\mathrm{m}$ totaling an area of $0.540\,\mathrm{mm}^2$. Based on the post-layout model, energy consumption was re-evaluated and found to be around 30 to 40% higher than for the topographical gate-level model.

D. Use case energy assessment

In the following, we assess the energy efficiency of open-loop and iterative linear MIMO detection. The results of this analysis are shown in Table V for precisions as in Table IV and full precision as reference. As predicted, energy consumption drops when switching to lower order constellations with lower precision requirements. When comparing the open-loop variant with iterative detection, the power consumed by the former is notably higher. This can be explained by the higher precision requirements in *it-0* and by the fact that the values themselves start converging in *it-1*, which reduces switching activity. Due to the shorter runtime, the energy consumption of open-loop detection is still less than for the iterative variant, though.

It shall be noted here that there is not just one correct precision and thus one energy number per use case. The precisions shown in Table IV are only necessary for the coded BERs to decay continuously down to the regions of 10^{-5} as in Fig. 10. Depending on the application, this kind of algorithmic performance is potentially not required when a few compromised bits do not diminish the overall user experience. As a result, it makes sense to trade energy savings for quality of service (QoS), e.g. in terms of coded BER, depending on the application requirements. Table VI illustrates this tradeoff for open-loop and iterative detection for 16QAM. It shows

	$\mathcal{P}_{ol}^{inv}$	$\mathcal{P}_{ol}^{mul}$	$\mathcal{P}_{ol}^{llr}$	$\mathcal{P}_{it}^{inv}$	$\mathcal{P}_{it}^{mul}$	$\mathcal{P}_{it}^{llr}$
4QAM	m8	m7	m4	m5	m4	m4
16QAM	m11	m10	m4	m8	m7	m4
64QAM	m12	m11	m4	m11	m10	m4

TABLE IV: Precision requirements for open-loop and iterative 4×4 MIMO detection

	4QAM		16QAM		64QAM		full prec.	
	mW	nJ	mW	nJ	mW	nJ	mW	nJ
it-0	119	23.8	134	26.8	138	27.6	143	28.6
it-1	86.9	24.3	105	29.4	117	32.8	124	34.7

TABLE V: Power and energy consumption of linear 4×4 MIMO detection on *s1m12e6* napCore

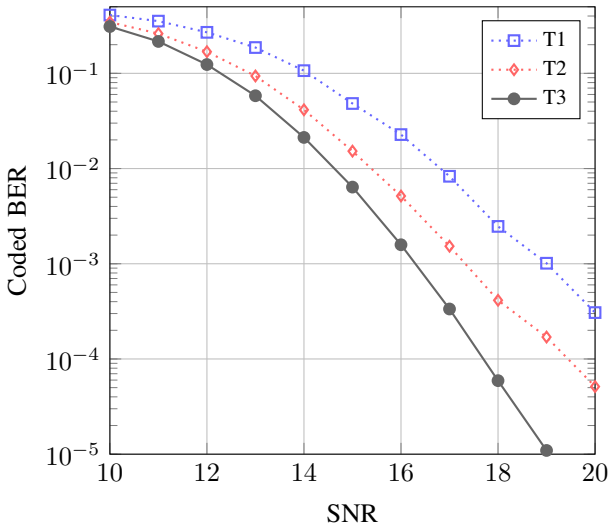

Fig. 12: Coded BER degradation with reduced numerical precision (16QAM, it-1)

the reduction of numerical precision requirements for coded BER targets T1, T2 and T3 in the range of 10^{-3}, 10^{-4} and 10^{-5} respectively. The complete trajectory of the coded BER for iterative detection subject to this precision reduction is shown in Fig. 12. It is interesting to see that already for a target coded BER of 10^{-4}, the mantissa width can be reduced from eight to five or less bits for iterative detection. This fact might potentially be exploited by integrating a set of small integer multipliers into the floating-point data path to handle operations with less precision requirements. The gain of such a measure can be approximated based on Fig. 8.

E. Efficiency comparison with state-of-the-art

To assess the efficiency of our design, we conduct a literature based comparison with state-of-the-art linear MMSE MIMO detectors for a 4×4 antenna configuration, which is summarized in Table VII. The implementation levels range from synthesized via layouted down to silicon implementations. Since linear detectors operate on symbol vectors, we define area and energy efficiency as processed symbol vectors per second and gate equivalent ($\mathrm{vec/s/GE}$) and processed symbol vectors per unit of energy ($\mathrm{vec/nJ}$), respectively. For comparability with [10], we consider energy consumption for the use of a 64QAM constellation. It is not straightforward how to include the silicon area of the memories into the comparison, since neither of the two linear detection algorithms requires caching in the vector memory. Also the application size of the linear MIMO detection algorithms is around 10% of the available program memory, which can be considered negligible compared to the rest of the design. Memory is therefore not included in the area comparison. Energy efficiency is given including memories, though.

Target BER	$\mathcal{P}^{inv}$	$\mathcal{P}^{mul}$	$\mathcal{P}^{llr}$	E [nJ]
Open-loop detection at 24 dB				
T1	m8	m7	m3	23.8
T2	m9	m8	m3	24.8
T3	m11	m10	m4	26.8
Iterative detection at 19 dB				
T1	m4	m3	m2	24.1
T2	m5	m4	m3	24.5
T3	m8	m7	m4	29.4

TABLE VI: QoS vs. energy, 4×4 MIMO detection (16QAM)

When assessing programmability and flexibility, different architectures can be characterized by their programming interface and data path reconfigurability. The tailored ASIC designs [14] and [10] both have no programming interface and are internally controlled by an FSM. The adaptive stream processing engine (ASPE) [4] is programmed in a VLIW-like fashion to configure the data flow through the functional units before stream processing begins. Even though an up-front configured data path supersedes accesses to the program memory during execution, it limits potential applications to those with a repetitive, regular data flow. Also, high throughput can only be guaranteed if a sufficient number of functional units is available to occupy the complete width of the processing pipeline. Therefore, the architecture can be considered moderately flexible. The remaining architectures sequentially reconfigure their data path for each instruction. In [15], an assembly-programmable array of multiply accumulate elements is shown, but the authors explicitly mention the limited flexibility of the data path. The reconfigurable ASIP (rASIP) design in [3] contains a two-dimensional coarse grained reconfigurable array (CGRA) configured by a RISC core via a configuration memory. While the RISC core can be easily programmed in assembly, the functional units of the CGRA have to be programmed by a configuration bitstream, which limits the ease of programming. Also the structure of the CGRA is highly tailored to the target application. As a result, the overall rASIP can be considered medium flexible and medium programmable. The *napCore*, programmable by scalar or SIMD assembly and equipped with a versatile instruction set, is clearly the simplest but also the most flexible and easy to use among the presented alternatives.

In terms of area efficiency, our design outperforms flexible designs [4] and [15] by a factor of three or four respectively. The comparison with [15] is particularly interesting, since both are floating-point designs with comparable hardware complexity. Nevertheless, our architectural measures described in Section II that maximize the utilization of the functional units within the vector data path result in a clear advantage in area efficiency. Despite the application specific, highly parallel structure of [3], our lightweight design still has a 27% higher area efficiency. The non-programmable ASIC solution in [14] is only superior by a marginal factor of 1.08, since it implements fixed-point matrix inversion by a series of Rank1 updates resulting in a comparably long runtime. The most efficient ASIC design [10] makes use of LU decomposition combined with a block floating-point number format, which allows a faster and more efficient matrix inversion. Hence, [10] is superior by a factor of 3 in terms of area efficiency. Energy efficiency is generally harder to compare since it depends on stimuli, test conditions, and the implementation level. Nevertheless, it should be mentioned that the tailored ASIC in [10] is 5.9 times more energy-efficient than our layout when operating at full precision for 64QAM. The gap decreases when switching to less dense constellations or trading QoS for energy efficiency.

	ASIC [14]	ASIC [10]	ASPE [4]	ARRAY [15]	rASIP [3]	napCore
Number format	fixed-pt.	block floating-pt.	fixed-pt.	floating-pt.	fixed-pt.	floating-pt.
Implementation	synth.	silicon	silicon	synth.	layout	layout
Matrix inversion algorithm	Rank1	LU	DnQ	DnQ	Rank1	DnQ
Iterative	no	yes	no	no	no	yes
Includes SINR calculation	no	yes	no	no	no	yes
CMOS technology [nm]	250	90	180	65	65	90
Clock frequency [MHz]	167	568	250	400	400	400
Area [kGE]	89	410	383	120	482	123
Area [mm^2]	N/A	1.5	3.7	N/A	1.4	0.54
Cycles per detection	102	18	83	204	17[a]	69 / 80 / 112[b]
Scaled clock frequency [MHz][c]	464	568	500	289	289	400
Energy efficiency [vec/nJ]	N/A	0.183[d]	N/A	N/A	N/A	0.031
Area efficiency [vec/s/GE]	51.1	86.3[d]	15.7	11.8	37.1[d]	47.1 / 40.7 / 29.0[b]

[a] One cycle added as opposed to [3] to complete equalization.
[b] Given for open-loop algorithm without and with SINR computation and iterative algorithm with SINR calculation.
[c] Clock frequency scaled linearly with feature size.
[d] Impact of LLR block and symbol/variance remapping was subtracted, since it is not part of the other architectures.

TABLE VII: Comparison with state-of-the-art 4×4 linear MIMO detectors

V. Conclusion

In this paper, we illustrated the potential as well as the limitations of a fully programmable floating-point processor core to compete with significantly less flexible architectures in terms of area and energy efficiency. We described a bundle of architectural measures that make our core a flexible and efficient target for algorithms based on complex-valued vector arithmetic. A versatile instruction set for complex vector arithmetic fosters high throughput. An optimized operand acquisition scheme including smart bypassing as well as vector arithmetic affine permutation units further improves the architectural throughput and hence the achieved area efficiency. Energy efficiency can be optimized by means of numerically aware processing for floating-point arithmetic, which allows the programmer to adapt the numerical precision at runtime to the application requirements to reduce switching activity and thereby energy consumption.

For a practical analysis, we chose 4×4 linear MIMO detection as a case study. We partitioned the detection algorithms into distinct sections with different numerical precision requirements, which again are different for each modulation scheme. Based on this analysis, we presented the resulting energy consumption using a post-layout model, where we saw the potential of numerically aware processing in conjunction with floating-point arithmetic. As an example, the relaxed precision requirements of the less dense 4QAM constellation allow the reduction of the used wordwidth and thereby the energy per MIMO detection by 17% (open-loop) or 30% (iterative) as opposed to full precision. We also showed how numerical precision and energy efficiency can be traded gradually for quality of service. An exemplary raise of the target coded bit error rate from 10^{-5} to 10^{-4} enabled a 17% decrease in energy consumption for iterative detection.

Compared to tailored MIMO detector implementations that provide less programmability and flexibility, we still found our design to be more efficient. Non-programmable ASIC implementations like [14] and [10] provide superior efficiency, but the difference is less than one order of magnitude. Our main contribution is therefore the reduction of the ASIC-ASIP efficiency gap while maintaining a high degree of flexibility and programmability.

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 802 LAN/MAN Standards Committee, Tech. Rep., Oct. 2009.

[2] J. Zyren, "Overview of the 3GPP Long Term Evolution Physical Layer," Freescale Semiconductor, Tech. Rep., Jul. 2007.

[3] X. C. et al., "Flexible, Efficient Multi-Mode MIMO Detection using reconfigurable ASIP," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 2014.

[4] S. Eberli, D. Cescato, and W. Fichtner, "Divide-and-Conquer Matrix Inversion for Linear MMSE Detection in SDR MIMO Receivers," in *NORCHIP, 2008.*, Nov. 2008, pp. 162–167.

[5] D. Guenther, T. Kempf, and G. Ascheid, "Numerical Aspects of MIMO OFDM PHY Layer Applications on SDR Platforms," *The Journal of Signal Processing Systems*, Dec. 2013.

[6] D. Guenther, R. Leupers, and G. Ascheid, "Mapping of MIMO Receiver Algorithms onto Application-Specific Multi-Core Platforms," in *Wireless Communication Systems (ISWCS 2013), Proceedings of the Tenth International Symposium on*, Aug. 2013.

[7] J. Han and M. Orshansky, "Approximate computing: An emerging paradigm for energy-efficient design," in *Test Symposium (ETS), 2013 18th IEEE European*, 2013, pp. 1–6.

[8] W. H. P. et al., *Numerical Recepies - The Art of Scientific Computing*, 3rd ed. Cambridge University Press, 2007.

[9] X. Wang and H. V. Poor, "Iterative (turbo) soft interference cancellation and decoding for coded CDMA," *Communications, IEEE Transactions on*, vol. 47, no. 7, pp. 1046–1061, Jul. 1999.

[10] 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*, vol. 46, no. 7, pp. 1754–1765, Jul. 2011.

[11] A. T. et al., "A Low Complexity Turbo MMSE Receiver for W-LAN MIMO Systems," in *Communications, 2006. ICC '06. IEEE International Conference on*, vol. 9, Jun. 2006, pp. 4119–4124.

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

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

[14] A. B. et al., "Algorithm and VLSI architecture for linear MMSE detection in MIMO-OFDM systems," in *Circuits and Systems, 2006. ISCAS 2006. Proceedings. 2006 IEEE International Symposium on*, May 2006, pp. 4–8.

[15] J. Eilert, D. Wu, and D. Liu, "Implementation of a programmable linear MMSE detector for MIMO-OFDM," in *Acoustics, Speech and Signal Processing, 2008. ICASSP 2008. IEEE International Conference on*, Mar. 2008, pp. 5396–5399.

Daniel Guenther received the Dipl.-Ing. degree in Electrical Engineering (Communications Eng.) from RWTH Aachen University in 2011. He is a Ph.D. student at the Institute for Integrated Signal Processing at RWTH Aachen University. His research focuses on software defined radio solutions for mobile, wireless communications.

Rainer Leupers received the M.Sc. and Ph.D. degrees in Computer Science with honors from the University of Dortmund, Germany, in 1992 and 1997. From 1997-2001 he was the chief engineer at the Embedded Systems chair at the University of Dortmund. During 1999-2001 he was also a team leader at ICD, where he headed industrial service projects. In 2002, Dr. Leupers joined RWTH Aachen University as a professor for Software for Systems on Silicon. His research and teaching activities comprise software development tools, processor architectures, and electronic design automation for embedded systems, with emphasis on compilers, ASIPs, and MPSoC design tools. He has been a co-founder of LISATek, an EDA tool provider for embedded processor design, acquired by CoWare Inc. He has served as consultant for various companies, as an expert for the European Commission in FP7, and in the management boards of large scale projects like UMIC, HiPEAC, ARTIST, SHAPES and Euretile.

Gerd Ascheid received his Diploma and Ph.D. (Dr.-Ing.) degrees in Electrical Engineering (Communication Eng.) from RWTH Aachen University. In 1988 he started as a co-founder and managing director CADIS GmbH which successfully brought the system simulation tool COSSAP to the market. In 1994 CADIS was acquired by Synopsys Inc., a California-based EDA market leader. From 1994-2003 Gerd Ascheid worked as Director/Senior Director with Synopsys. In 2003 Gerd Ascheid joined RWTH Aachen University as a full professor in the Institute for Communication Technologies and Embedded Systems. His research interest is in wireless communication algorithms, application specific integrated platforms, in particular, for mobile terminals and cyber physical devices. Gerd Ascheid has co-authored three books, published numerous papers in the domain of digital communication algorithms and ASIC implementation and is founder of several successful start-up companies.