A 283 pJ/b 240 Mb/s Floating-Point Baseband Accelerator for Massive MU-MIMO in 22FDX

Oscar Castañeda, Luca Benini, and Christoph Studer

Department of Information Technology and Electrical Engineering, ETH Zurich, Zurich, Switzerland

***Abstract*—We present PULPO, a floating-point baseband-processing accelerator for massive multi-user multiple-input multiple-output (MU-MIMO) basestations (BSs). PULPO accelerates matrix-vector products, not only with a matrix but also with its Hermitian, as well as affine transforms and nonlinear projections used in iterative algorithms that outclass traditional linear methods in various applications. PULPO is integrated in a system-on-chip (SoC) with a tight integration to the system's data memory, facilitating data exchange and co-operation with 8 RISC-V cores. The fabricated accelerator achieves comparable efficiency as recently-proposed fixed-point baseband processors, while eliminating the burdens associated with fixed-point design, thus simplifying massive MU-MIMO BS development.**

I. Introduction

Baseband processing for modern high-rate wireless communication systems poses significant implementation challenges in terms of power consumption and throughput. Emerging technologies, such as massive multi-user multiple-input multiple-output (MU-MIMO) [1] and millimeter-wave (mmWave) communication [2], further aggravate the situation as they require processing of high-dimensional signals acquired at hundreds of basestation (BS) antennas at rates exceeding billions of tasks per second. As a result, baseband processing at infrastructure BSs for such systems is expected to be carried out with application-specific integrated circuits (ASICs), which achieve the best energy efficiency and highest throughput. Over the last few years, a wide range of baseband-processing ASICs for massive MU-MIMO BSs have been proposed; see, e.g., [3]–[6].

A. Programmable Accelerators for Massive MU-MIMO

Baseband-processing ASICs often implement a single algorithm that is highly optimized for specific system parameters and operation conditions. Specialization limits their adaptability to time-varying system and channel conditions, which was shown to be key to low-power mmWave massive MU-MIMO baseband processing [6]. Furthermore, supporting an entirely different set of system parameters (e.g., the number of BS antennas) might even require fabrication of another ASIC. To counter these issues, flexible and programmable hardware accelerators have emerged recently [7], [8]. The work in [7] proposes a configurable systolic array comprising 64 processing elements (PEs), while [8] proposes an application-specific instruction-set processor (ASIP) comprising 8 lanes; both of these designs use complex-valued 32-bit fixed-point datapaths. These two accelerators provide improved flexibility and adaptability over an ASIC, and improved hardware efficiency over a general processor. However, to support vastly different conditions (such as system dimensions, propagation conditions, or modulation and coding schemes) with fixed-point datapaths, extensive simulations and re-configuration/re-programming are necessary to make best use of the available dynamic range. In fact, failing to re-adjust a fixed-point datapath to a given scenario (often significantly) degrades performance; see, e.g., Sec. IV-A.

B. Contributions

We propose PULPO, a programmable floating-point baseband-processing accelerator for massive MU-MIMO BSs. PULPO supports a wide range of baseband algorithms for both the uplink and the downlink. By implementing a (transprecision) floating-point datapath, PULPO adapts to different system conditions without requiring fixed-point parameter tuning or a specialized compiler to compensate for changes in numeric representations. PULPO is tightly integrated with the data memory of a system-on-chip (SoC) containing 8 Parallel Ultra-Low Power (PULP) RISC-V cores; the SoC's architecture follows that of [9]. This tight integration enables PULPO to access the SoC's entire data memory at full bandwidth, which facilitates communication and interaction with the RISC-V cores. Hardware measurements of the 22 nm FD-SOI prototype demonstrate that PULPO offers comparable performance and efficiency to the fixed-point solutions in [7], [8], while further improving flexibility and adaptability.

II. The PULPO Baseband-Processing Accelerator

PULPO supports several baseband algorithms for both uplink and downlink in massive MU-MIMO BSs equipped with B antennas serving U single-antenna user equipments (UEs).

A. Supported Baseband Algorithms

1) Linear Algorithms: In the uplink, the BS uses the received signals to estimate the transmitted UE symbols. To minimize complexity, one typically resorts to the linear minimum mean-squared error (LMMSE) equalizer, which simply corresponds to a matrix-vector product (MVP). In the downlink, the BS must generate a precoded vector that removes MU interference in the UE signals to be transmitted [10]. This task can be achieved with the linear zero-forcing (ZF) precoder, also applied as an MVP. Computing the LMMSE and ZF matrices requires MVPs too. Thus, MVPs are crucial tasks to be accelerated by PULPO.

This work was supported in part by ComSenTer, one of six centers in JUMP, a SRC program sponsored by DARPA. The work of CS was also supported by an ETH Research Grant and by the US NSF under grants CNS-1717559 and ECCS-1824379. Contact author: O. Castañeda (e-mail: caoscar@ethz.ch)

The authors thank A. Di Mauro, M. Scherer, M. Eggiman, G. Rutishauser, F. Glaser, and Microelectronics Design Center for assistance with chip fabrication.

ESSCIRC 2022 - IEEE 48th European Solid State Circuits Conference (ESSCIRC) | 978-1-6654-8494-7/22/$31.00 ©2022 IEEE | DOI: 10.1109/ESSCIRC55480.2022.9911311

2) Nonlinear Algorithms: Linear detection and precoding algorithms are suboptimal, especially (i) in systems with a large load factor U/B [11], (ii) for BSs that utilize low-resolution digital-to-analog converters (DACs) [12], or (iii) for channels with short coherence times [13]. Thus, PULPO also supports (among many others) the following nonlinear algorithms that alleviate these issues: (i) box-constrained (BOX) equalization achieving near-maximum-likelihood performance even in systems with large load factors [14]; (ii) biconvex x-bit precoding (CxPO) for BSs with low-resolution DACs [12]; and (iii) projection-onto-convex-hull (PrOX) for joint channel estimation and data detection (JED) [13]. PULPO further supports computing finite-alphabet equalization and precoding matrices, which enable low-power linear detection and precoding [15].

B. Operating Principle

All of the above linear and nonlinear baseband algorithms involve one or more of the following three atomic operations:

$$\mathbf{z} = \mathbf{A}\mathbf{x} + \mathbf{y}\ (\mathcal{M}),\ \mathbf{z} = \mathbf{y} - \tau\mathbf{A}^H\mathbf{x}\ (\mathcal{H}),\ \mathbf{z} = \text{prox}(\rho\mathbf{x})\ (\mathcal{P}).$$

Here, $\mathbf{A}$, $\mathbf{x}$, $\mathbf{y}$, τ, and ρ are algorithm-specific quantities. For example, LMMSE equalization only needs an MVP $(\mathcal{M})$ with $\mathbf{A}$ being the equalization matrix, $\mathbf{x}$ the signals received by the BS, and $\mathbf{y} = \mathbf{0}$. The C1PO and PrOX algorithms iterate MVPs $(\mathcal{M})$ followed by projection $(\mathcal{P})$, resulting in iterations of the form $\mathbf{x}^{(t+1)} = \text{prox}(\rho\mathbf{A}\mathbf{x}^{(t)})$; see [12], [13] for the specific choices for ρ, $\mathbf{A}$, and $\mathbf{x}^{(0)}$. The BOX and C2PO algorithms iteratively perform all three operations $(\mathcal{M})$, $(\mathcal{H})$, and $(\mathcal{P})$ as follows: $\mathbf{x}^{(t+1)} = \text{prox}(\rho(\mathbf{x}^{(t)} - \tau\mathbf{A}^H(\mathbf{A}\mathbf{x}^{(t)})))$, where ρ and τ are system-dependent tunable parameters. For all of these algorithms, the prox$(\cdot)$ operator is an element-wise clipper.

PULPO accelerates exactly the atomic operations $(\mathcal{M})$, $(\mathcal{H})$, and $(\mathcal{P})$ for any problem dimension that fits the available memory. To achieve this goal at the highest efficiency, one must support (i) MVPs with a matrix $\mathbf{A}$ and also its Hermitian $\mathbf{A}^H$ without requiring explicit transposition of $\mathbf{A}$ in memory, (ii) hardware-support for sequencing these atomic operations for iterative algorithms and for varying system dimensions, and (iii) flexibility for different prox$(\cdot)$ operators. We next detail a VLSI architecture that satisfies all of these requirements.

III. VLSI Architecture

Fig. 1(a) illustrates the SoC comprising 8 PULP RISC-V cores with 128 kB of data memory that is shared with the PULPO baseband-processing accelerator. PULPO consists of 16 PEs connected in a unidirectional ring network. As detailed in Fig. 1(b), each PULPO PE is a configurable complex-valued floating-point multiply-accumulate unit equipped with a projection unit that is able to perform the most common prox$(\cdot)$ operators. The PE's complex-valued multiplier is pipelined, and can be configured to conjugate one of the operands or to compute two real-valued multiplications in parallel.

A. Cannon's Algorithm for MVPs

The PEs implement Cannon's algorithm [16] to efficiently calculate MVPs: Each PE is associated with a different row of the matrix $\mathbf{A} \in \mathbb{C}^{N\times N}$, and all PEs operate simultaneously on a different entry of the vector $\mathbf{x} \in \mathbb{C}^N$ (and thus on a different column of their respective row). The PEs then circularly exchange their vector $\mathbf{x}$ entries (blue path in Fig. 1) to complete the MVP $\mathbf{A}\mathbf{x}$ in $\mathcal{O}(N)$ clock cycles. The advantage of Cannon's algorithm is that it can also be used to multiply a vector $\tilde{\mathbf{x}} \in \mathbb{C}^N$ by the Hermitian $\mathbf{A}^H$ with the same matrix readout pattern, thus avoiding the need of rewriting the memory to explicitly transpose $\mathbf{A}$; this is achieved with minimal overhead by the PEs circularly exchanging the partial products $a^*_{j,i}\tilde{x}_j$ (red path in Fig. 1) instead of the entries of the vector $\tilde{\mathbf{x}}$.

B. Higher-Dimensional Problems and Memory Access

Since PULPO incorporates 16 PEs, it natively executes MVPs with 16×16 matrices. To support larger square and non-square matrices, we divide them in blocks of 16×16. MVPs with wider matrices are computed by accumulation via the $\mathbf{y}$ vector in $(\mathcal{M})$ and $(\mathcal{H})$; MVPs with taller matrices are simply computed in a time-multiplexed manner. Intermediate matrix dimensions are handled through zero-padding.

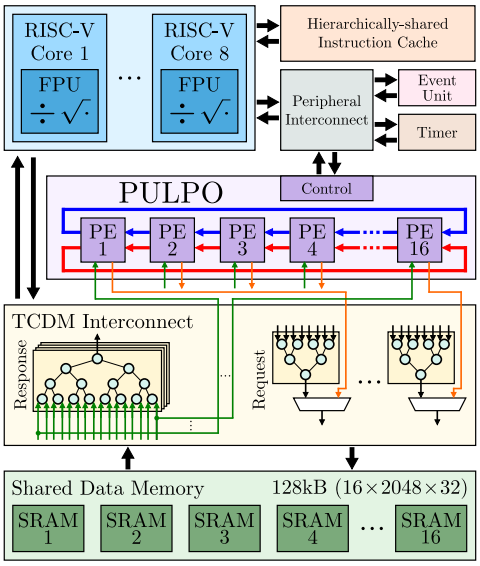
To perform tasks with matrices as large as the SoC's memory can support, each PULPO PE is tightly-coupled to a memory bank (SRAM) of the shared data memory: Each PULPO PE has a read/write connection to a specific SRAM that bypasses the tightly-coupled-data-memory (TCDM) interconnect used to route communication between RISC-V cores and data memory (green and orange paths in Fig. 1). To avoid interfering with the cores' memory accesses, PULPO's requests (orange paths) have low priority and will only reach a memory bank if no core is trying to access that bank. To reduce memory requests, PULPO has an internal memory that stores four 16×16 blocks.

C. Operating PULPO

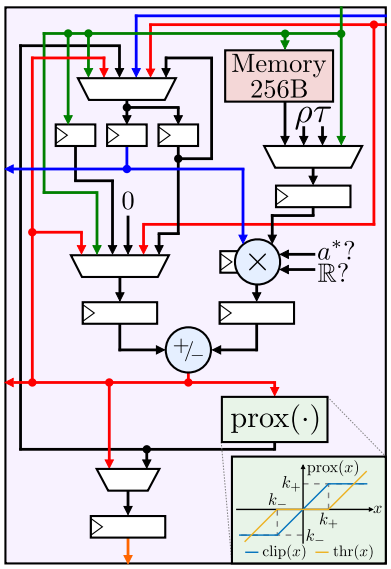
Tightly coupling PULPO with the data memory requires the operands $\mathbf{A}$, $\mathbf{x}$, and $\mathbf{y}$ to be aligned with the memory banks. Furthermore, to simplify readout logic for PULPO, we store the matrix $\mathbf{A}$ in a so-called *skewed* fashion (see Fig. 1(c)). These two data-arrangement requirements are handled by the RISC-V cores when receiving or generating the input operands. Once the operands have been arranged in memory, the cores can configure PULPO's control unit with their dimension and memory location (among other parameters; see Fig. 1(d)), and start PULPO's operation. PULPO offers hardware support to automatically and independently sequence through the $(\mathcal{M})$, $(\mathcal{H})$, and $(\mathcal{P})$ atomic operations, and iterate over such sequence. Specifically, PULPO can iterate over $(\mathcal{M})$ or $(\mathcal{H})$ only, as well as over the sequences $(\mathcal{M})\rightarrow(\mathcal{P})$, $(\mathcal{H})\rightarrow(\mathcal{P})$, and $(\mathcal{M})\rightarrow(\mathcal{H})\rightarrow(\mathcal{P})$, where the output $\mathbf{z}$ of each sequence step becomes the input $\mathbf{x}$ of the subsequent step. Other sequences can be implemented by using the RISC-V cores to execute the $(\mathcal{M})$, $(\mathcal{H})$, and $(\mathcal{P})$ operations in the desired order. Once finished with its task, PULPO raises an event and the cores can access the results directly from the shared data memory.

D. Exploiting Shared Memory with the RISC-V Cores

The shared data memory enables tight interaction between the RISC-V cores and PULPO. For example, PULPO's prox$(\cdot)$



(a) PULPO-SoC system overview



(b) PULPO processing element (PE)

Bank 1	(1,1)	(1,2)	(1,3)
Bank 2	(2,2)	(2,3)	(2,1)
Bank 3	(3,3)	(3,1)	(3,2)

(c) Skewed 3×3 matrix $\mathbf{A}$

off.	function	off.	function
00	$\mathbf{x}$ address	1C	$\rho\&\tau$ address
04	$\mathbf{y}$ address	20	k_-, k_+
08	$\mathbf{A}$ address	24	# iterations
0C	$\mathbf{z}$ address	28	dimensions
10	ρ and fix $\rho\&\tau$		and op mode
		2C	stop
14	τ	30	resume
18	τ in use	3C	start

(d) PULPO configuration words

floating-point format	number of bits			
	sgn	exp	mant	tot
binary16	1	5	10	16
bfloat16	1	8	7	16
binary8	1	5	2	8
hardware	1	8	10	19

(e) PULPO floating-point formats

Fig. 1. Implementation details of the PULPO baseband-processing accelerator and its host SoC.

unit only supports element-wise clipping operations (required by the nonlinear algorithms from Sec. II-A2) and thresholding operations (useful, e.g., for sparse signal recovery and channel vector denoising). For other tasks, PULPO can rely on the RISC-V cores to perform any kind of projection. To this end, PULPO simply writes intermediate results to memory, raises an event, and waits to receive a resume command after the cores have calculated the projection. Another use case is early stopping: In each iteration, PULPO writes the current iterate to memory and raises an event so that the cores can keep track of the iterates, e.g., to stop PULPO upon algorithm convergence.

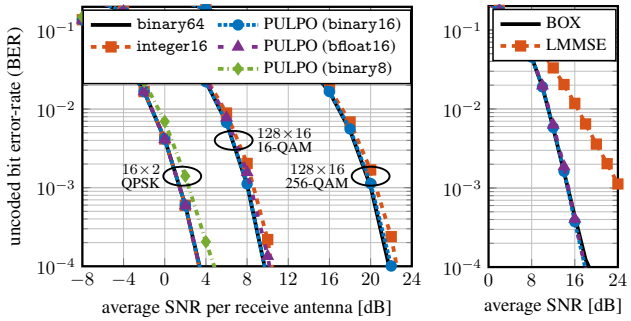
E. Configurable Floating-Point Precision

A key feature is PULPO's transprecision support for different floating-point formats: `binary16`, `bfloat16`, and `binary8`. To improve area efficiency, all formats share 19-bit arithmetic units supporting the maximum number of required mantissa and exponent bits (`hardware`; see Fig. 1(e)). When operating with `binary8`, PULPO only needs to access half of the memory banks. Moreover, PULPO has a real-valued mode that performs two real-valued MVPs (same matrix, different vectors) at once. The RISC-V cores' floating-point units (FPUs) additionally support `binary32` with the architecture details provided in [17]. The RISC-V cores' FPUs can also perform division and square-root operations, further extending the capabilities of PULPO.

IV. Implementation Results

A. Bit Error-Rate (BER) Performance

Fig. 2(a) shows the uncoded BER performance of the implemented PULPO for LMMSE equalization with varying system configurations. For a fair comparison, we consider a design similar to that of [8], which uses 16-bit fixed-point (`integer16`) representation per real and imaginary parts, as well as 24-bit accumulation. To illustrate the advantages of floating-point support, we perform the following experiment: We calibrate the



(a) LMMSE (b) 16×16, QPSK

Fig. 2. Uncoded bit-error rate (BER) vs. SNR performance of PULPO when operating with (a) LMMSE equalization and (b) BOX equalization. In this second plot, the black and red curves correspond to BOX and LMMSE with `binary64`; the other curves follow the same legend as in (a).

position of the fixed-point of `integer16` so that it achieves an acceptable BER across a wide variety of system configurations. Then, we evaluate the performance using that single fixed-point format for different system configurations. As shown in Fig. 2(a), having such a "universal" fixed-point configuration causes a performance loss of up to 0.5 dB at 0.1% BER. While a fixed-point design can recover from this performance loss, doing so would require (i) extensive simulations and (ii) to re-configure/re-program the accelerator/ASIP to shift the fixed points accordingly. In contrast, PULPO's quarter-precision (`binary16`) floating-point format achieves virtually the same performance as double-precision (`binary64`) floating-point for all system configurations, without requiring any manual re-calibration. Fig. 2(a) also shows that, as we reduce the system dimensions or modulation scheme, alternative floating-point formats with fewer mantissa bits achieve comparable performance: For a 16×2 ($B \times U$) system, halving the floating-point bitwidth (`binary8`) results in a 1 dB loss at 0.1% BER.

Fig. 2(b) exemplifies how nonlinear algorithms can outperform linear algorithms—here, we consider a system with

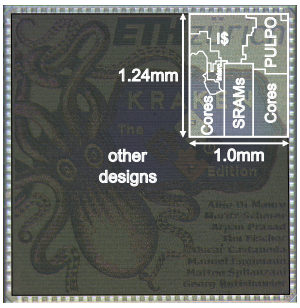


Fig. 3. Chip micrograph; the implemented PULPO-SoC is highlighted.

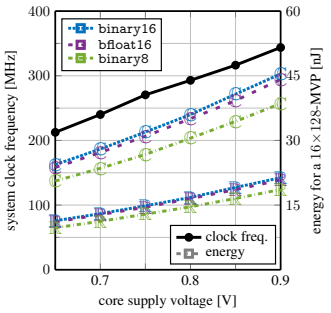


Fig. 4. Measured system clock frequency and energy vs. core supply.

TABLE I
COMPARISON WITH OTHER MASSIVE MU-MIMO ACCELERATORS

	PULPO	Prabhu [8]	Peng [7]	Tang [3]	Jeon [4]	Wen [5]	Castañeda [6]
Programmable B, U, alg.	**yes**	**yes**	**yes**	no	no	no	no
Floating-point support	**hw**	sw	no	no	no	no	no
Shared core memory	**yes**	no	no	no	no	no	no
BS antennas B	128	128	128	128	256	256	32
UEs U	16	8	8	32	32	32	16
Technology [nm]	22	28	28	40	28	40	65
Core supply [V]	0.8	1.0	0.9	0.9	0.9	1.1	1.0
Core area [mm^2]	1.24	2.2	4.8	0.58	0.37	0.73	4.20
Max. frequency [MHz]	293	290	800	425	400	290	312
Total power [mW]	97	180	528	221	151	87	396
Throughput [Gb/s]	0.240	0.169	1.54	2.76	0.354	1.96	9.98
Accelerator area [mm^2]	0.42	0.38	4.8	0.58	0.37	0.73	2.41
Accelerator power [mW]	68	91.7	528	221	151	87	290
Area eff.[a] [Gb/s/mm^2]	0.57	0.91	0.66	28.6	1.97	16.1	106
Energy[a] [pJ/b]	283	214	167	19.1	208	7.10	2.13

[a]Technology normalized to 22 nm and 0.8 V core supply.

a load factor $U/B = 1$ and obtain a 9 dB improvement at 0.1% BER. Similar examples can be found for massive MU-MIMO BS architectures with low-resolution DACs [12] or low-resolution equalizers [15]. We reiterate that PULPO meets double-precision performance and its native acceleration of $(\mathcal{M})$, $(\mathcal{H})$, and $(\mathcal{P})$ achieves an 11× speed-up compared to execution on the 8 RISC-V cores with their FPUs.

B. Measurements and Comparison

Fig. 3 shows the fabricated 9 mm^2 chip with the presented SoC highlighted; the rest of the chip contains other designs. The PULPO-SoC occupies an area of 1.24 mm^2, with the PULPO accelerator, the data memory, and the 8 RISC-V cores taking 18%, 16%, and 32% of the area, respectively. At 0.8 V supply voltage and 300 K, the SoC achieves a maximum clock frequency of 293 MHz, with the critical path being between the instruction cache (I$) and a RISC-V core. When considering a 128-antenna BS communicating with 16 UEs using 256-QAM, PULPO achieves a maximum detection/precoding throughput of 240 Mb/s. For applications in which the cores are not using their FPUs, PULPO is responsible for 70% of the SoC's power.

Fig. 4 shows the frequency and energy versus the SoC's supply voltage. We consider PULPO's energy per complex-valued 16×128 MVP ($\mathbb{C}$ markers) or per real-valued 16×128 MVP ($\mathbb{R}$ markers) for different floating-point formats. We observe that `binary16` dissipates the most energy, caused by the largest amount of mantissa bits. The alternative 16-bit format, `bfloat16`, requires 3% lower energy, while `binary8` saves 16%. Two real-valued MVPs (same matrix, two vectors) are 7% more energy efficient than one complex-valued MVP.

Tbl. I compares PULPO to the other programmable massive MU-MIMO accelerators [7], [8], as well as to existing detection ASICs [3]–[6]. Since PULPO handles all critical baseband-processing operations on its own, we only take into account the area and energy used by PULPO and the shared data memory when reporting area- and energy-efficiency. After technology normalization, the three programmable accelerators are orders-of-magnitude less efficient than the ASICs [3]–[6], which is the cost of enabling programmability and adaptability. Nonetheless, PULPO's area- and energy-efficiency is only 1.6× and 1.7× worse, respectively, than the other two accelerators [7], [8], which is a low price to pay for the flexibility and ease-of-use provided by native floating-point support.

V. CONCLUSIONS

We have presented PULPO, the first programmable floating-point baseband accelerator for massive MU-MIMO BSs. Our accelerator is tightly integrated with the data memory of an 8-core RISC-V SoC, and supports a wide range of algorithms. Our comparison with recent fixed-point baseband accelerators [7], [8] has revealed that PULPO achieves comparable throughput and energy efficiency while avoiding fixed-point optimizations and simplifying design, thus resolving the common misconception that floating-point baseband processing is inefficient.

REFERENCES

[1] E. G. Larsson *et al.*, "Massive MIMO for next generation wireless systems," *IEEE Commun. Mag.*, vol. 52, no. 2, pp. 186–195, Feb. 2014.
[2] T. S. Rappaport *et al.*, *Millimeter Wave Wireless Communications*, 2015.
[3] W. Tang *et al.*, "A 0.58mm^2 2.76 Gb/s 79.8pJ/b 256-QAM massive MIMO message-passing detector," in *IEEE Symp. VLSI C.*, Jun. 2016.
[4] C. Jeon *et al.*, "A 354 Mb/s 0.37 mm^2 151 mW 32-user 256-QAM near-MAP soft-input soft-output massive MU-MIMO data detector in 28nm CMOS," *IEEE SSC Lett.*, vol. 2, no. 9, pp. 127–130, Sep. 2019.
[5] C.-C. Wen *et al.*, "A 1.96Gb/s massive MU-MIMO detector for next-generation cellular systems," in *IEEE Symp. VLSI C.*, Jun. 2020.
[6] O. Castañeda *et al.*, "A resolution-adaptive 8 mm^2 9.98 Gb/s 39.7 pJ/b 32-antenna all-digital spatial equalizer for mmWave massive MU-MIMO in 65nm CMOS," in *IEEE ESSCIRC*, Sep. 2021, pp. 247–250.
[7] G. Peng *et al.*, "A 2.92-Gb/s/W and 0.43-Gb/s/MG flexible and scalable CGRA-based baseband processor for massive MIMO detection," *IEEE JSSCC*, vol. 55, no. 2, pp. 505–519, Nov. 2019.
[8] H. Prabhu *et al.*, "A 1070pJ/b 169Mb/s quad-core digital baseband SoC for distributed and cooperative massive MIMO in 28 nm FD-SOI," in *IEEE Symp. VLSI C.*, Jun. 2021.
[9] D. Rossi *et al.*, "Vega: A ten-core SoC for IoT endnodes with DNN acceleration and cognitive wake-up from MRAM-based state-retentive sleep mode," *IEEE JSSCC*, vol. 57, no. 1, pp. 127–139, Jan. 2022.
[10] S. Jacobsson *et al.*, "Quantized precoding for massive MU-MIMO," *IEEE Trans. Commun.*, vol. 65, no. 11, pp. 4670–4684, Nov. 2017.
[11] M. Wu *et al.*, "Large-scale MIMO detection for 3GPP LTE: Algorithms and FPGA implementations," *IEEE JSTSP*, pp. 916–929, Oct. 2014.
[12] O. Castañeda *et al.*, "1-bit massive MU-MIMO precoding in VLSI," *IEEE JETCAS*, vol. 7, no. 4, pp. 508–522, Dec. 2017.
[13] ——, "VLSI designs for joint channel estimation and data detection in large SIMO wireless systems," *IEEE TCAS-I*, pp. 1120–1132, Mar. 2018.
[14] C. Jeon *et al.*, "Mismatched data detection in massive MU-MIMO," *IEEE TSP*, vol. 69, pp. 6071–6082, Oct. 2021.
[15] O. Castañeda *et al.*, "Finite-alphabet MMSE equalization for all-digital massive MU-MIMO mmWave communication," *IEEE JSAC*, Sep. 2020.
[16] L. Cannon, "A cellular computer to implement the Kalman filter algorithm," Ph.D. dissertation, Montana State University, 1969.
[17] S. Mach *et al.*, "FPnew: An open-source multiformat floating-point unit architecture for energy-proportional transprecision computing," *IEEE TVLSI*, vol. 29, no. 4, pp. 774–787, Dec. 2020.