Optimizing QAM Demodulation with NEON SIMD and Algorithmic Approximation Techniques

Ilias Papalamprou[1][0009−0008−5731−770X], Giorgos Armeniakos[1][0000−0001−7184−3740], Ioannis Stratakos[1][0000−0002−1103−8202], George Lentaris[1,2][0000−0003−1664−8648], and Dimitrios Soudris[1][0000−0002−6930−6847]

[1] National Technical University of Athens
[2] University of West Attica
{ipapalambrou, armeniakos, istratak, dsoudris}@microlab.ntua.gr
glentaris@uniwa.gr

Abstract. In any telecommunication system, it is crucial to have a high-performance receiver to meet the desired requirements. However with the newer protocols demanding high order constellations, the demodulation process in the receiver becomes a bottleneck. To facilitate the implementation of telecommunication systems on embedded platforms, in this work we explore optimizations to the QAM demodulation, by applying SIMD operations with the NEON engine along with algorithmic approximation techniques. We implement a NEON-based Demodulator using the Approximate LLR algorithm, while we also propose an approximate method for QAM16/QAM64 that focuses on one quadrature for calculating the required Euclidean distances, along with the respective NEON accelerator. We perform a trade-off analysis between system's BER and execution time of the Demodulator and the receiver module for the base and approximate implementations, while also exploring the impact of different bit widths and precision in computations. We demonstrate that our approximate technique can achieve ×18–×37 speedup over the original algorithm without BER deviations on uncoded channels, while the use of LDPC is also examined.

Keywords: SIMD · NEON · QAM · Demodulation · Approximation.

1 Introduction

In most digital telecommunication systems, hardware-based processing blocks within the Physical (PHY) Layer play a critical role, directly impacting overall system performance and reliability [6]. Recognizing their inherent limitations and methods to enhance both functionality and performance is crucial. This necessity becomes more pronounced with the increasing adoption of Software Defined Radio (SDR) technology across various applications [1].

SDR systems replace traditional hardware components with software and/or hardware/software implementations, offering enhanced flexibility and adaptability through easily modifiable radio behavior via software updates. A representative example of such a framework is the open-source GNU Radio [3], which includes various signal processing blocks capable of emulating different communication protocols. While its primary focus is on testing telecommunication scenarios in realistic communication links, it lacks on software optimizations that might lead to increased throughput, rate or performance.

Most of the recent research works focus on optimizing fundamental modules, such as: 1) Transmitter (TX), 2) Channel, and 3) Receiver (RX), with great focus given on the RX, since it demands the highest computational resources, owing to the intricate Digital Signal Processing (DSP) algorithms employed for data processing. Specifically, the RX is responsible for reversing the TX operation on the received signal transmitted over a noisy channel, involving demodulation and decoding to recover the original data. Among these tasks, demodulation of received symbols stands out as one of the most computationally intensive processes in the RX, alongside Channel Decoding. Indicatively, as we demonstrate in a software-based telecommunication processing pipeline with QAM64 and LDPC Decoder, the Demodulator block accounts for 20%–30% of the total execution time in RX, making it one of the bottlenecks for achieving the high throughput required for several scenarios (e.g., satellite communications). While several research efforts have focused on enhancing RX's execution time through hardware acceleration features of CPUs, such as Intel's AVX512 Extensions [7] and algorithmic optimizations in various demodulation algorithms [11, 13], none have applied optimizations across different layers of abstraction.

To this end, through this work, we examine and combine for the first time both algorithmic optimizations and software acceleration techniques in telecommunication applications. Specifically, targeting embedded ARM-based devices, we implement an accelerator for the Approximate LLR algorithm, utilizing Single Instruction Multiple Data (SIMD) operations provided by the NEON engine [12]. Furthermore, we propose a novel approximate algorithm for the LLR calculation in the Demodulator for both QAM16 and QAM64 schemes. This algorithm focuses solely on one quadrature, thus reducing the required computations while still maintaining almost the same output quality compared to the conventional method. Finally, by using different bit precisions in the computations, we examine different trade-offs between the execution time and the overall accuracy of the system. Our evaluation shows that the proposed algorithmic approximation along with its NEON-based implementation, achieves $\times 2$–$\times 4$ speedup depending on the modulation type, over the original NEON-based LLR algorithm. Additionally, the approximate NEON demodulator exhibits identical Bit Error Rate (BER) performance for uncoded data, while the scenario of using encoded data (e.x. using LDPC) is also examined.

2 Related Work

In the field of software based digital communication systems, extensive research has been conducted in recent years. Authors in paper [5] proposed a software optimized digital communication transceiver for the DVB-S2 protocol, that leverages parallel processing as well as Single Instruction Multiple Data (SIMD) operations of general-purpose processors. This implementation is based on the telecommunication library AFF3CT [4], which also utilizes multi-threaded and multi-core execution, as well as task level pipelining for the different transceiver modules. Other works like [16], focus on ultra low power and resource constrained devices, where an Internet-of-Things (IoT) oriented SDR platform was introduced. Software-based computations were implemented for the selected protocols (LoRa and Sigfox) on the ARM Cortex-M4 microcontroller, along with a hardware-based module for the RF front-end. Extensive research has been conducted on creating a software-based testbed for designing latest generation network protocols. The methodologies outlined in papers [9, 8] introduce a framework capable of supporting Single-Input Single-Output (SISO) and Multiple-Input Multiple-Output (MIMO) communication schemes at over-the-air interface. These approaches involve accelerating multiple algorithms on servers, through the utilization of SIMD techniques Intel CPUs provide.

Regardless of the extensive software optimizations for telecommunication systems, the continuous scale of network protocols keep increasing the demand for higher computational resources. Specifically, in order to meet performance requirements, constellation dimensions are also scaling, with some reaching as high as QAM4096. However this scaling increases the complexity and the computational resources required to implement the algorithms [15]. In particular, the Demodulator in the RX, due to use of high order constellations introduces a large overhead in processing the received symbols. Therefore, over the years various approximation techniques have been explored to optimize the demodulation process by reducing the required arithmetic operations. For example, authors in [11] proposed a simplified approach for calculating the LLR in 8-PSK and 16-PSK schemes. Their work reduces the needed calculations by dividing the constellation into smaller segments and calculating the LLR values there, since it requires fewer adjacent symbols. Researchers in [13] also focused on the demodulation of APSK constellations and proposed a method with decreased complexity. They are based on the piecewise form of the LLR calculation, in order to compute the number of required Euclidean distances in a limited region.

The approaches, although aiming at more efficient demodulation techniques, they only simulate the accuracy of the algorithms without optimizing them for deployment in an embedded system. On the other hand, our work distinguishes and presents a cross-layer optimization approach, providing a software accelerated demodulation implementation, that targets ARM-based processors equipped with NEON technology, along with a novel approximate QAM demodulation method. Furthermore, our work is protocol-independent and can be effective for both terrestial and non-terrestial networks that support these modulation schemes.

3 QAM Demodulation

In digital communication systems, QAM modulation involves converting a binary sequence into the complex plane (I, Q), where 'I' represents the in-phase component and 'Q' represents the quadrature component of the transmitted signal. Upon receiving the signal, the receiver must reverse this transformation, a process known as *Demodulation*. This operation is responsible for undoing the modulation process. Next, we will briefly discuss the two main categories of demodulation algorithms.

- ***Hard-Decision*** algorithms determine the reference constellation symbol with the minimum distance from the received symbol. Upon finding this reference symbol, its binary code is then employed to demodulate the received symbol.
- ***Soft-Decision*** algorithms, on the other hand, do not produce a set of binary values as a result. Instead, their operation revolves around generating a metric known as the Log-Likelihood Ratio (LLR). This metric is calculated for each bit in the binary code of every symbol and represents the logarithm of the probability that '0' was transmitted over to the probability that '1' was transmitted for a received symbol.

In this work, our focus centers in the field of *Soft-Decision* algorithms. The initial LLR algorithm, known as Exact LLR (abbreviated as ELLR thereafter), assumes equal probability for all received symbols, and calculates the LLR value in an AWGN channel [10] using Eq. 1. Given the computational complexity of logarithmic and exponential operations, researchers have sought simpler alternative forms, leading to the development of Approximate LLR (referred to as ALLR). Unlike the ELLR equation, which considers all constellation points, the computation of the ALLR is simplified. It involves using only the nearest constellation point to the received sample, considering either '0' or '1' at that specific bit position. As defined in [10], the ALLR of bit b is given by Eq. 2.

$$L_b = \log\left[\frac{\Pr(b=0|r)}{\Pr(b=1|r)}\right] = \ln\left\{\frac{\sum_{s\in S_0} e^{-\frac{1}{\sigma^2}[(x-s_x)^2+(y-s_y)^2]}}{\sum_{s\in S_1} e^{-\frac{1}{\sigma^2}[(x-s_x)^2+(y-s_y)^2]}}\right\} \tag{1}$$

$$L_b = -\frac{1}{\sigma^2}\left\{\max_{s\in S_1}\left[(x-s_x)^2+(y-s_y)^2\right] - \max_{s\in S_0}\left[(x-s_x)^2+(y-s_y)^2\right]\right\} \tag{2}$$

where (x, y) are the received (I, Q) symbol coordinates, $s_{x,y}$ is the reference constellation symbol and σ^2 is the noise variance. For the rest of this paper we consider and examine the ALLR demodulation based on Eq. 2 as our target algorithm for acceleration.

3.1 SIMD-based ALLR

Examining Eq. 2 indicates the possibility of parallelization capabilities, enabling the calculation of multiple arithmetic operations per LLR value. On ARM-based

embedded devices, the NEON engine emerges as a promising solution to accelerate applications. NEON is a general–purpose SIMD engine, that extents the ARM instruction set. It works seamlessly with its own independent datapath pipeline and register file and is able to perform packed SIMD processing that act upon vectorized data (it's registers are vectors of elements of the same data type). This capability is that we aim to leverage in this study to compute multiple arithmetic operations in parallel. Furthermore, we introduce a generic procedure suitable for various modulations, including QAM16/64/256/512/1024, with no customization needed per modulation.

In Fig. 1 we present the original NEON-based ALLR demodulator we developed. In order to ensure that we have a uniform range of results across multiple data ranges, we normalize the computed Euclidean distances, using the Euclidean distance of the input symbol from the first point of the constellation. Furthermore, our NEON-based accelerator is designed to support different I/Q input widths (8/16-bit), as well as varying precision (16/32-bit) for the multiplication with the constant α ($= -1/\sigma^2$). Moreover, the 16-bit multiplication is implemented using the `vqdmulhq_n_s16()` intrinsic since it can handles overflows internally [2], while for the multiplication with 32-bit of precision we implemented a custom function to deal with it (Fig. 1(b)). Specifically, we overcome possible overflows by following these steps: 1) split a single input vector into two vector with half the size 2) perform the multiplication in each new vector separately with 32-bit precision, and 3) convert the result back to 16-bit by selecting which bits to keep. Furthermore, to find the maximum of each Euclidean distance a maximum-tree structure was developed, that uses a single intrinsic to obtain the maximum of a vector, while also using previously calculated comparison results to reduce the complexity.

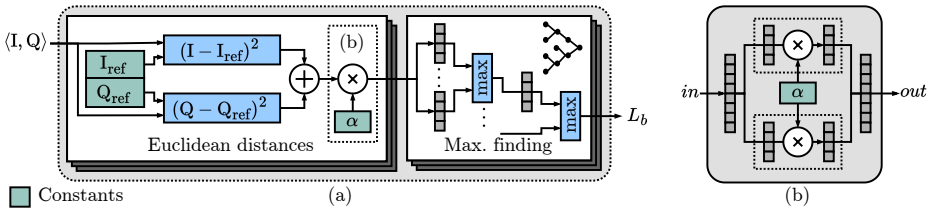


Fig. 1: (a) ALLR NEON Demodulator; (b) Custom multiplication module.

Next, we present an initial evaluation of our NEON-ALLR demodulator for all combinations of input widths and multiplication precision for various QAM modulations.

3.2 Initial Exploration

The execution time as well as the BER performance for each Demodulator configuration is examined, with the details for the setup listed in Section 4. From Fig. 2

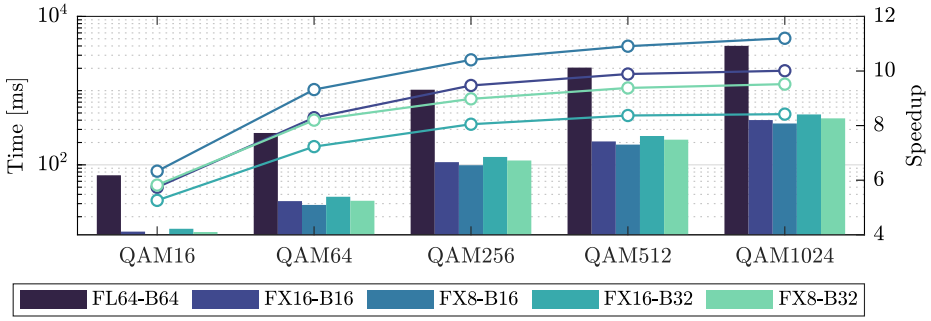


Fig. 2: Execution Time performance of ALLR NEON Demodulator.

we can observe the acceleration using NEON over the original floating-point implementation (denoted as FL64-B64). We note that for better visualization of the data due to the large execution time difference between the constellations, the left y-axis is in logarithmic scale. By observing the right y-axis plot lines that refer to the speedup of each NEON implementation over the baseline model, we notice that as the QAM constellation increases, the use of the NEON engine becomes more efficient, resulting in an exponential increase in speedup until a saturation point is reached. While a speedup of up to $\times 11$ was achieved, further performance gains were explored by applying algorithmic optimizations along with SIMD acceleration to the original Approximate LLR algorithm.

3.3 Proposed Approximate QAM Demodulation

Additional optimizations were applied to the original Eq. 2, in order to reduce the number of required arithmetic operations with support for QAM16/64 constellations. Specifically, in the proposed approach we calculate only the Euclidean distances on the quadrature the received symbol belongs. Then, knowing the Gray code used in the constellation, we can define the '1' and '0' in each bit position, in order to find the maximum Euclidean distance in each of the two groups, which now contain 1/4 of the elements than those in the original algorithm. For bits located in two specific digit positions (LSB-1 & MSB for QAM16, LSB-3 & MSB for QAM64) in any quadrature mapped with Gray code, we observe that there is only '1' or '0' as a possible bit value, thus $s \notin S_0$ or $s \notin S_1$ respectively. Therefore if we refer back to Eq. 2, one of the two sets for finding the maximum Euclidean distance is empty ($S_0 = \emptyset$ or $S_1 = \emptyset$). In our preliminary approach, in order to avoid this indeterminate form we relied only on the available distances and assumed that the maximum Euclidean distance of the empty set is a constant value c. Even though we evaluated different values for c, including $c = 0$ and $c = -\infty$ (where $-\infty$ refers to a large negative number), this technique resulted in significantly elevated LLR Mean Relative Error (MRE) values and a noticeably deviated BER. Therefore for the cases where $S_0 = \emptyset$ or $S_1 = \emptyset$, a different methodology had to be applied to replace the

Table 1: Demodulation arithmetic operations per symbol

Module	Operation	QAM16		QAM64	
		Base	Approx.	Base	Approx.
Euclidean distance	Mul.	48	18	192	54
	Add./Sub.	48	18	192	54
Max. finding	Comparisons	42	16	186	64

maximum value of non-existing elements. As a solution we utilized two additional Euclidean distances $d(s, s_c)$, one for each scenario: whether none '1' or none '0' are present. The supplementary calculations are performed by selecting a predetermined symbol from the neighboring quadrature with coordinates $(s_{x,c}, s_{y,c})$. After exploring various combinations, we identified that selecting the symbol closest to the center of the axis from the diagonally adjacent quadrature performs with the least MRE. We note that different methods could possibly be applied, like selecting each time randomly or in a cyclic pattern the symbol. Eq. 3 describes the above mentioned procedure for calculating the LLR $L_{b,apx}$ where b is the current bit position and Q_i is the quadrature the received symbol belongs, with $i = 1, \ldots, 4$. Furthermore, Table 1 shows the computational complexity difference over the original ALLR calculation.

$$L_{b,apx} = L_{b,s\in Q_i} + \begin{cases} 0 & S_0 \neq \emptyset \text{ and } S_1 \neq \emptyset \\ -\frac{1}{\sigma^2}[(s - s_{x,c})^2 + (s - s_{y,c})^2] & S_0 = \emptyset \text{ or } S_1 = \emptyset \end{cases} \quad (3)$$

NEON Implementation: An accelerator for the proposed approximate demodulation algorithm is developed for NEON, with the architecture shown in Fig. 3. In order to efficiently use SIMD operations, the reference Euclidean distances used for normalization are calculated in parallel for multiple symbols. Additionally for the implementation of the smaller QAM16 constellation, I/Q coordinates are packed in a single vector, while this is complemented by the according access pattern in the array that contains the reference I/Q values (Fig. 3(b)). Similarly to the NEON Demodulator analyzed in Section 3.1, multiple implementations were examined regarding bit precision. Specifically, 16/32-bit multiplication precision is supported using the same module of Fig. 1. Regarding the input bit width, for the QAM16 constellation due to the small number of parallelizable calculations with SIMD instructions, the 8-bit vectors are not used efficiently, therefore we only examined 16-bit input. For the larger QAM64 constellations however we implemented both 8-bit and 16-bit input width. Hence in total we have two different configurations for QAM16 and four different configurations for QAM64. Furthermore, the Maximum finding of the Euclidean distances reuses the aforementioned tree-based structure, tailored for the new dimensions of the sets.

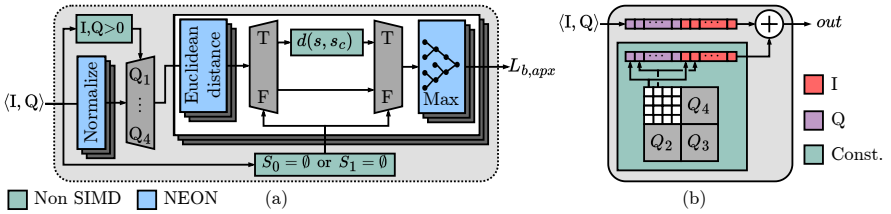


Fig. 3: (a) Proposed NEON demodulation; (b) Access Pattern for I and Q values.

4 Evaluation

This section presents the experimental results of the different Demodulator implementations, focusing on the execution time as well as the BER performance. The functions of the telecommunication processing pipeline are developed in C++, while the experiments were conducted on the Zynq Ultrascale+ MPSoC ZCU106 evaluation board, which features a quad-core ARM A53 processor clocked at 1GHz and 4GB of DDR4 RAM. Regarding the operating system we opted for Ubuntu 20.04.6 LTS (aarch64) with kernel 5.4.0-1015-xilinx-zynqmp, while for code compilation we relied on GCC (g++ version 9.4.0). All the subsequent implementations were compiled using the `-O3 -march=native` flag. The experimental telecommunication processing pipeline used is presented in Fig. 4, where an AWGN channel model is used. We examined the performance of the RX in two configurations: 1) *Without Forward Error Correction (FEC)*: Uncoded transmission and the classification of the output bit b_i was performed by comparing the sign of the calculated LLR value: $b_i = 0, \text{if } LLR < 0$ or $b_i = 1, \text{if } LLR \geq 0$, and 2) *With FEC*: LDPC for encoding/decoding with a fixed coding rate of $3/4$ through all simulation runs. We also note that each simulation run consists of 32 blocks with 64800 symbols in each one, with the execution time calculated from taking the average of the blocks.

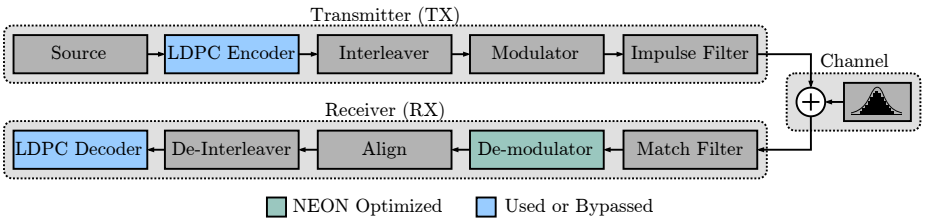


Fig. 4: Experimental telecommunication Chain

Table 2 shows the execution time of the NEON-based Demodulator as well as the whole RX in the different configurations, where we can observe the impact on acceleration when using our approximate approach. Although the 16-16 com-

bination outperforms the other configurations our selected implementation is the 16-32, since it still offers sufficient acceleration gains and as we will analyze, it exceeds the others in terms of BER.

Table 2: NEON Demodulator Execution Time (ms)

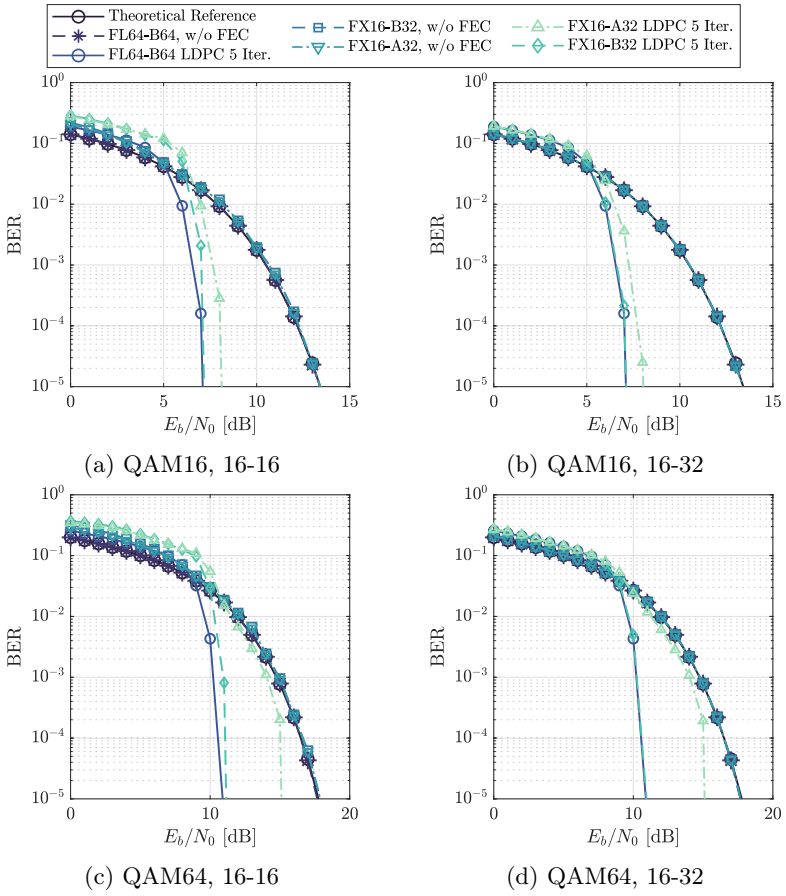
Conf.*	**QAM16**						**QAM64**					
	Demod.		RX-I†		RX-II‡		Demod.		RX-I†		RX-II‡	
	Base	Apprx.	Base	Apprx.	Base	Apprx.	Base	Apprx.	Base	Apprx.	Base	Apprx.
16-16	10.81	3.91	16.84	9.94	522.88	531.72	32.60	5.95	41.81	15.17	861.87	798.06
16-32	10.89	3.86	16.91	9.87	553.99	548.33	32.68	6.96	41.88	16.18	855.58	814.48
8-16	8.93	-	14.96	-	538.55	-	26.07	6.41	35.29	15.63	814.73	779.13
8-32	9.64	-	15.67	-	533.46	-	28.88	7.10	38.10	16.32	836.33	832.87

* I-M: I: Input bits; M: Mul. precision † RX w/o FEC. ‡ RX w/ LDPC at 5 iter.

In Fig. 5 the BER results of the most representative configurations are presented. The 8-bit input width configurations are not included, since they offer comparable speedup but worse BER performance. For comparing the base and approximate NEON demodulation, we used the default floating point implementation, while we also compared the performance along the theoretical BER values (denoted as "Theoretical Reference") in an AWGN channel for uncoded data, which are calculated using equations stated in [14].

In an uncoded channel, regarding both the QAM16 and QAM64 constellations, the NEON-based ALLR and the proposed approximate Demodulator closely follow the BER curve of the floating-point model as well as the theoretical estimation, while the 16-32 configuration achieves the best BER performance. When using LDPC encoder/decoder for the base QAM16/QAM64 demodulation, the NEON-based ALLR shows similar BER with the base floating point model, while the best results are obtained with the 16-32 configuration. The proposed approximation, when using LDPC presents deviations in the BER curves compared to the baseline model. Specifically the QAM64 performs worse as opposed to the smaller QAM16. Upon analysis of the relative error of the approximated LLRs compared to the original values, the loss of accuracy occurs due to the points where an additional Euclidean distance from an adjacent quadrature is required.

Fig. 6 shows the execution time of the Demodulator and the RX for the two most representative configurations (16-16 and 16-32) regarding accuracy versus acceleration trade off. Due to the sufficient acceleration and best BER performance, the 16-32 configuration was finally selected. Additionally we note that for all listed results, the higher speedup is observed for QAM64. Referring to Fig 6a, 6d where the execution time of only the Demodulator is presented, for the NEON-based ALLR we observe a speedup of $\times 6.41$–$\times 7.92$ over the original floating-point implementation. Furthermore the proposed approximation approach features a $\times 18.11$–$\times 37.19$ acceleration over the baseline implementation, thus improving the execution time of the base algorithm in NEON by a factor of

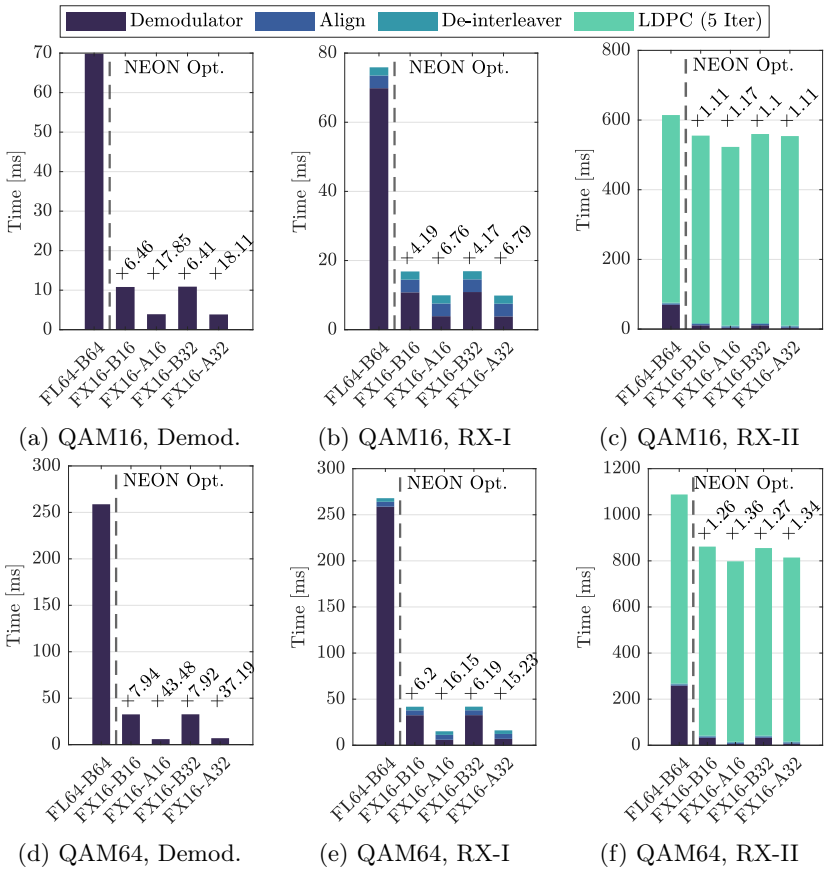


(a) QAM16, 16-16 (b) QAM16, 16-32

(c) QAM64, 16-16 (d) QAM64, 16-32

Fig. 5: BER Analysis

×2.83–×4.7. Afterwards we examine the effect of the accelerated Demodulator to the whole RX system, where apart from the LDPC decoder we also include the execution time of the de-interleaver (in row by column configuration) and the alignment block. In Fig. 6b, 6e where we have uncoded data, the Demodulator occupies most of the execution time and the proposed approximate Demodulator improves the RX's execution time by a factor of ×6.79–×15.23, while the ALLR in NEON obtains a speedup of ×4.17–×6.19. When using LDPC decoder at 5 iterations, as seen in Fig. 6c and Fig. 6f, the decoder bottlenecks the RX's performance. The impact of this block is significant, to the extend that the NEON-ALLR achieves speedup of ×1.1–×1.27, while the proposed approximate Demodulator achieves acceleration of ×1.11–×1.34 over the baseline model. Without loss of generality and as a proof of concept we consider 5 iterations for the LDPC decoder, since after conducting offline exploration and evaluating dif-



(a) QAM16, Demod. (b) QAM16, RX-I (c) QAM16, RX-II

(d) QAM64, Demod. (e) QAM64, RX-I (f) QAM64, RX-II

Fig. 6: Execution time for different configurations

ferent iteration settings, the BER performance was sufficient. However, we note that for a more robust system, the number of iterations might be increased.

5 Conclusions

This work presented an accelerator for the QAM Demodulation function utilizing SIMD operations provided by the NEON engine featured in ARM processors. Two different algorithms are utilized: a baseline LLR soft-demodulation algorithm along with a proposed approximate approach of the same algorithm. Furthermore various configurations are presented for each accelerator, regarding the input bit-width and the multiplication precision. We examine the different Demodulator implementations based on their execution time in a telecommunication processing pipeline with or without FEC, while we explore the accuracy performance throught the BER metric. Our findings reveal that our proposed approximation demodulation method for QAM16/QAM64 is promising, with

×19–×39 acceleration over the base floating-point model and no accuracy loss in uncoded data. When using LDPC BER deviations are higher, with QAM16 performing the best. To extend the approximation approach for larger constellations, we suggest using more Euclidean distances from the adjacent quadrature.

Acknowledgments. This work was partially supported by European Union H2020 project PRIVATEER (grant agreement 101096110).

References

1. Akeela, R., Dezfouli, B.: Software-defined radios: Architecture, state-of-the-art, and challenges. Computer Communications **128**, 106–125 (2018)
2. ARM: Arm neon intrinsics guide. https://developer.arm.com/architectures/instruction-sets/intrinsics, [Accessed 25-03-2024]
3. Blossom, E.: Gnu radio: tools for exploring the radio frequency spectrum. Linux journal **2004**(122), 4 (2004)
4. Cassagne, A., Hartmann, O., Leonardon, M., He, K., Leroux, C., Tajan, R., Aumage, O., Barthou, D., Tonnellier, T., Pignoly, V., et al.: Aff3ct: A fast forward error correction toolbox! SoftwareX **10**, 100345 (2019)
5. Cassagne, A., Leonardon, M., Tajan, R., Leroux, C., Jégo, C., Aumage, O., Barthou, D.: A flexible and portable real-time dvb-s2 transceiver using multicore and simd cpus. In: 2021 11th International Symposium on Topics in Coding (ISTC). pp. 1–5. IEEE (2021)
6. Chamola, V., Patra, S., Kumar, N., Guizani, M.: Fpga for 5g: Re-configurable hardware for next generation communication. IEEE Wireless Communications **27**(3), 140–147 (2020)
7. Cornea, M.: Intel avx-512 instructions and their use in the implementation of math functions. Intel Corporation pp. 1–20 (2015)
8. Filo, M., Xia, Y., Nikitopoulos, K.: Saccess: Towards a software acceleration framework for 5g radio access networks. In: 2021 IEEE International Mediterranean Conference on Communications and Networking (MeditCom). pp. 318–323. IEEE (2021)
9. Georgis, G., Filo, M., Thanos, A., Husmann, C., Ducoing, J.D.L., Tafazolli, R., Nikitopoulos, K.: Sword: Towards a soft and open radio design for rapid development, profiling, validation and testing. IEEE Access **7**, 186017–186040 (2019)
10. Hamkins, J.: Performance of low-density parity-check coded modulation. In: 2010 IEEE Aerospace Conference. pp. 1–14. IEEE (2010)
11. Olivatto, V.B., Lopes, R.R., de Lima, E.R.: Simplified llr calculation for dvb-s2 ldpc decoder. In: 2015 IEEE International Conference on Communication, Networks and Satellite (COMNESTAT). pp. 26–31. IEEE (2015)
12. Reddy, V.G.: Neon technology introduction. ARM Corporation **4**(1), 1–33 (2008)
13. Sandell, M., Tosato, F., Ismail, A.: Efficient demodulation of general apsk constellations. IEEE Signal Processing Letters **23**(6), 868–872 (2016)
14. SIMONS, M., Alouini, M.: Digital communication over fading channels: a unified approach to performance analysis (2000)
15. Singya, P.K., Shaik, P., Kumar, N., Bhatia, V., Alouini, M.S.: A survey on higher-order qam constellations: Technical challenges, recent advances, and future trends. IEEE Open Journal of the Communications Society **2**, 617–655 (2021)
16. Xhonneux, M., Louveaux, J., Bol, D.: A sub-mw cortex-m4 microcontroller design for iot software-defined radios. IEEE Open Journal of Circuits and Systems (2023)