IEEE Open Journal of the
Industrial Electronics Society

Received 15 September 2022; revised 13 October 2022; accepted 26 October 2022. Date of publication 7 November 2022;
date of current version 23 November 2022. The review of this article was arranged by Associate Editor C.-M. Chituc.

Digital Object Identifier 10.1109/OJIES.2022.3220048

High Performance Implementation of Next Generation Aeronautical Communication Systems

S. KURZ [1], E. GRINGINGER [1], C. RIHACEK [1], AND T. SAUTER [2,3] (Fellow, IEEE)

[1]Frequentis AG, 1100 Vienna, Austria
[2]Institute of Computer Technology, TU Wien, 1040 Vienna, Austria
[3]Department for Integrated Sensor Systems, University of Continuing Education, 3500 Krems, Austria

CORRESPONDING AUTHOR: S. KURZ (E-mail: stefan.kurz@frequentis.com).

ABSTRACT Current air traffic communication systems are mainly based on voice communication, with a newer digital communication technology called L-band digital aeronautical communication system (LDACS) being investigated in the single European sky air traffic management research project. An essential feature of this communication infrastructure is the encoding of data that guarantees reliable transport. While the encoding in the transmission path is straightforward, the decoding is computationally expensive due to the peculiarities of the convolutional codes used. As the target platform for the communication equipment is an embedded system, a proper system design is essential for the receiving path to ensure real-time processing. This article therefore focuses on the hardware/software codesign of the functional system parts needed for decoding in the LDACS radio receiver. We describe the fundamental design considerations, followed by the actual implementation in form of software and field-programmable gate array based hardware modules. Subsequently, the decoding solution was verified to prove a standard-compliant system. In an experimental validation, the actual system was fed with test data from a reference system. This allows conclusions to be drawn about system characteristics like data throughput, latency, and error correction. The resulting system demonstrates high-performance decoding that can exceed the desired requirements for quality and speed for use in the LDACS communication system.

INDEX TERMS Aeronautical communication, embedded systems, hardware acceleration, hardware-based decoding, hardware-software codesign, L-band digital aeronautical communication system (LDACS).

I. INTRODUCTION

Today's air traffic communication systems are mainly based on voice communication. This is surprising at first sight, but understandable when considering that the foundation of these systems was formulated as early as 1949 in Annex 10 of the convention on international civil aviation organization (ICAO) [1]. Inevitably, this means that existing solutions have deficiencies that are either challenging or even impossible to resolve. The most striking characteristic is that still today, transmission of information such as flight data from air traffic control to an aircraft is mainly voice based. Therefore, the exchange of even the simplest data such as flight altitude or speed by means of spoken language take an enormous amount of time compared to what could be possible with digital-based systems common in our daily lives. Considering the increasing number of flights in the future, the currently used systems will therefore rather sooner than later reach their capacity limits [2].

In the future, aircrafts will be able to connect to high-bandwidth communications when at the airport terminal. But as of today, once they get into the air, they must continue to rely on narrow-band radio channels that are limited to a data throughput capacity measured in the low kilo-bits per second range. For voice communications, both air-to-ground and air-to-air links are via broadcast channels without any authentication, encryption, or other embedded protective measures that are common in modern systems. Both voice and data communications equipment are mostly ground-based using

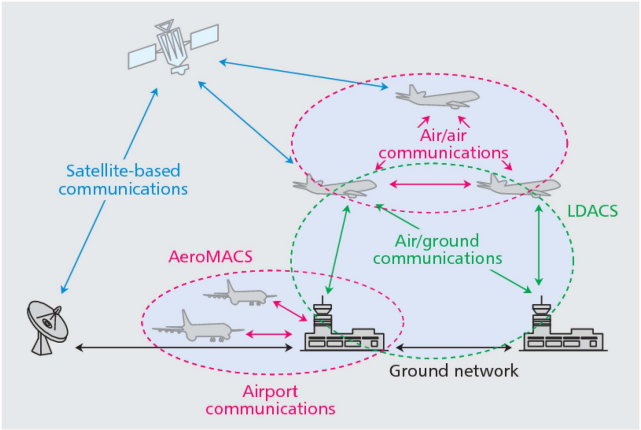

FIG. 1. Aeronautical communication infrastructure [6].

the high frequency (HF) or very high frequency (VHF) frequency band. Due to the lack of available spectrum, existing HF and VHF data links (VHF digital link (VDL) Mode 2) [3] are incapable of supporting broadband services now or in the future. Technology vendors are developing enhanced capabilities for satellite-based communications. However, satellite technologies are not able to cover the high bandwidth demands of heavily congested flight routes and airports. They are also dependent on weather conditions and introduce transmission delays that would be problematic for some applications.

For all of these reasons, modernization of air traffic management and its communications is necessary, and possible candidates for future air traffic management (ATM) communication systems have been studied to provide the basis for the development of future aviation communications infrastructure [4]. In the course of this, the SESAR (single European sky air traffic management research) program on behalf of the European commission addresses the modernization of European air traffic management [2]. Within this context, research projects have been awarded in the areas of communications, navigation, productivity enhancement, optimization, and more. It is agreed within the regulatory and research communities that a single communications infrastructure will not be sufficient for all phases of a flight. Therefore, as shown in Fig. 1, the future communications infrastructure will consist of several digital connectivity technologies integrated into a communication network. To ensure future communication between air traffic controllers and pilots, the digital communication technology called L-band digital aeronautical communication system (LDACS) is being investigated [5]. In addition to LDACS, which serves as a continental communications path for air-to-ground and air-to-air communications, other technologies are planned for noncontinental operations using satellites and a data link for large ground airport scenarios [6].

The difficulties in the development and implementation of aeronautical communication systems are mainly related to the conditions of the operating environment in combination with the long range of the systems [7], [8]. When a ground station communicates with an aircraft, the high speed of the aircraft results in a nonnegligible frequency shift due to the Doppler effect. In addition, communication must be ensured both in flight and on the ground, e.g., near the airport. This means that such a communication system must be able to handle high speeds with direct line-of-sight between the ground station and the aircraft, but must also function in very diffuse and wave-reflective environments on the ground.

To meet these very demanding requirements, special attention must be paid to the coding used in the system. The transmitted data must be encoded such that as many errors as possible that occur at the receiver can be corrected, while still achieving the highest possible effective data throughput [9]. This requires an elaborate encoding procedure which has been foreseen in the LDACS standard. The downside is that the associated decoding tasks are typically computationally intense and require an effective implementation strategy. The particular problem in LDACS is that the protocol leaves very little time for completing the decoding task because an immediate response is required in the next communication frame. The question therefore is how to ensure such stringent real-time processing, especially when LDACS is implemented on an embedded platform, which is the typical case for communication equipment. A pure and straightforward software implementation likely does not meet the performance requirements. Implementing the decoder in hardware to accelerate data processing is an obvious solution, but may not be efficient in terms of resource usage and implementation effort. This article proposes a mixed implementation of the decoding branch of LDACS on an embedded platform, where some decoder blocks are implemented in hardware, while others are kept in software. To that end, we investigate a hardware/software codesign approach in the sense of optimal design partitioning. We evaluate the pros and cons of hardware versus software implementation for each block of the decoding chain individually and also with respect to the overall system design. Besides the fundamental concepts, this article also presents the first functional decoder implementation of the upcoming standard, together with performance results related to the real-time implementability of the overall system.

The rest of this article is organized as follows. Section II discusses the LDACS system overview, Section III presents related work on decoding blocks and hardware/software partitioning. Section IV develops the essential design considerations, whereas Section V describes the actual implementation and test environment. Section VI presents the performance results. Finally, Section VII concludes this article with an outlook to future work.

II. LDACS SYSTEM OVERVIEW

The reference for the technical details of the system were obtained from the current version of the LDACS system specification [10]. Given the long innovation cycles in aeronautical technologies, the development goal of LDACS was to devise a future-proof system [11]. To that end, LDACS

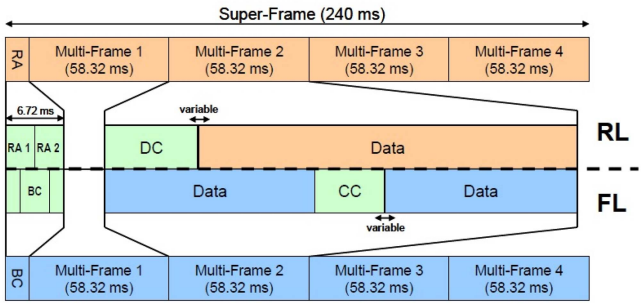

FIG. 2. **LDACS super-frame structure [10].**

was designed as a bidirectional digital communication system organized in cells with a cell radius of up to 200 nautical miles or 370 km [8]. In addition to regular digital data communications, LDACS was also designed for digital voice communications to support the current standard voice-based communications method used by air traffic control.

A. PHYSICAL LAYER AND FRAME STRUCTURE

The physical layer specification of LDACS utilizes the so-called L-band, which is the frequency range between 1 and 2 GHz [12]. An LDACS channel enabling a connection between a ground station and all airplanes in its associated cell uses a bandwidth of 500 kHz. The data to be transmitted is therefore packed into frames with a duration of 240 ms, called super-frames, with one channel used to transmit information from the ground to the aircraft, called forward link, and another for the opposite direction, called reverse link [7]. Each super-frame in turn can be split into the starting broadcast or random-access section followed by four multiframes consisting of control information (DC, CC) and actual data, depicted in Fig. 2.

Depending on the current environmental situation, the radio channel in which the actual transmission takes place can be fundamentally different, and the technical implementation must be able to handle those different situations. It is therefore based on adaptive coding mechanisms agreed between the LDACS ground station and each air station. A data stream to be sent is modulated using either quadrature phase-shift keying (QPSK) or quadrature amplitude modulation (QAM) at 16 or 64 b per symbol. This defines how data are converted into complex wave forms leading to the frequency components in an orthogonal frequency-division multiplex (OFDM) symbol [9]. These considerations determine the essential throughput requirement, with the combination of a 64QAM modulation and a coding rate of 3/4 requiring the highest overall decoder throughput of 2.22 Mb/s, which is the minimum required to meet real-time processing capabilities.

In addition to the minimum throughput, another point regarding the LDACS framing structure shown in Fig. 2 must be considered. Since all airplanes share one link to the ground, there is a resource allocation mechanism in the control channel of the forward link holding the information which frame part in the reverse link can be utilized by a specific airplane.

FIG. 3. **LDACS decoding chain architecture.**

Since transmit and receive frame start are aligned in time, the information packed in one receive multiframe ideally leads to a response in the next transmit multiframe. This creates an additional processing speed requirement leading to a throughput of 2.26 up to 6.87 Mb/s, depending on the selected modulation scheme. In the event that this timing requirement cannot be met, it would be possible to apply this information in the subsequent multiframe, so that this requirement can be considered optional. Since digital voice should be transmitted via LDACS as well, latency is also a critical factor of the system. This total latency should not exceed 200 ms on the end-to-end link to guarantee audio quality that is subjectively perceived as good.

B. ERROR CORRECTION

Another system aspect in coping with varying transmission channel quality is error correction. In LDACS, this is realized by combining convolutional coding and Reed–Solomon coding together with interleaving in order to correct occurring errors most effectively. Fig. 3 illustrates the decoding system as a block diagram with all the necessary individual tasks. In the transmission system, data are encoded in packets, followed by processing at the physical layer, where OFDM symbols are generated from these data frames. After the addition of synchronization symbols, the symbols are converted into their associated analog signal and then modulated to a preselected LDACS frequency.

The receiving system part detects LDACS OFDM frames by searching for synchronization symbols that define the beginning of each frame. The symbols thus reconstructed are then converted into binary data by interpreting the received carrier signal within a symbol containing 2–6 b, depending on the modulation technique chosen. Since errors may have occurred on the transmission channel, i.e., on the transport medium on which the radio waves propagate from the transmitter to the receiver, decoding mechanisms are used to restore the originally transmitted data stream [10].

C. SYSTEM REQUIREMENTS AND IMPLEMENTATION OPTIONS

In summary, the desired system must satisfy two essential, firm requirements: 1) it must be able to handle a throughput of at least 2.22 Mb/s and optionally up to 6.87 Mb/s; 2) an end-to-end latency of 200 ms must not be exceeded to achieve the desired performance metric for digital voice.

Owing to the structure of the LDACS coding and decoding system, a software implementation is an obvious option. This was done as a baseline implementation in the form of a SystemC simulation model to assess performance and feasibility of the system specification. The SystemC implementation offers the basic possibility to use the code of the simulation

model with slight modifications also as actual coding software. However, as will be shown in later sections, it was found that high processing times, especially for decoding, are a problem for real-time implementation in pure software, creating the need for a more powerful system realization.

The performance problem can be avoided by moving the decoding tasks to hardware-based processing, such as by using a field-programmable gate array (FPGA). FPGAs are integrated circuits based on configurable logic blocks whose behavior is defined by their interconnection. This allows a circuit design that enables the implementation of tasks with very low processing times, exploiting possibilities for parallelization of algorithms or optimized computing structures. To meet the aforementioned system requirements, shifting computationally intensive decoding blocks and parts from pure software to hardware-based implementation appeared to be the most promising solution. This hardware/software codesign approach is discussed in more detail in the following sections.

III. RELATED WORK

Mixed hardware/software-based implementations of coding and decoding chains are widely used in modern communication systems because of the demanding requirements. Although indepth studies exist for LDACS, analyzing the concept of use together with system performance and requirements in general [11], there is no work that discusses the direct implementability of an LDACS system or parts of it. Nevertheless, there are parallelisms and analogies in different other applications of hardware/software codesign, specifically in the hardware/software partitioning of functional blocks.

As a first approach, implementations of computational intense decoding blocks was analyzed independently, such as convolutional decoding implementations [13] and the inversion of the Reed–Solomon code [14]. In these and other comparable articles, an implementation of these system parts in hardware-based form, e.g., in the form of an application-specific integrated circuit (ASIC) or FPGA, is very often proposed. However, the connection of these decoder blocks with others is not discussed.

Considering a hardware/software codesign approach, in [15], Abdallah et al. showed the applicability of this design method for increased efficiency of aerospace systems. The article explains the usage of codesign in an abstract way especially for safety and real-time critical applications. Since the focus is on the methods and not on the actual system implementation, the application of the concept to other use cases is difficult.

A research work by Chih-Hung et al. focuses on a low-cost rate control framework for video encoding. It concentrates on the control flow, specifically the exchange of control messages without the need for high data rates [16]. Unfortunately, direct exchange mechanisms for data streams between hard and software are missing, so that a straightforward applicability of the concept to LDACS is not possible and would require further investigation.

Considering codesign methods for communication systems, Bolsens et al. investigated the possible use of a design environment for heterogeneous systems combined in one chip [17]. The environment focuses on the encapsulation of individual tasks that are later implemented in hardware or software which communicate via dedicated control channels. The main disadvantage of their approach is that the proposed model is highly abstract and therefore such methods cannot be applied directly to dedicated implementations.

A more modern approach to hardware/software codesign for data-intensive applications is shown by Wiangtong et al., who use a special platform called Ultra-SONIC for their design ideas [18]. The platform allows a partial reconstruction of a hardware system in form of an FPGA and is therefore very adaptable. Despite its good applicability to many applications, this method does not allow hardware and software to exchange data at high rates, so that the results of the implementation cannot be applied to a decoding system directly.

Ibraheem et al. investigated an FPGA-based decoding of advanced audio coding (AAC) focusing on a fast and parallel processing resulting in an achieved decoding rate of max. 30 Mb/s [19]. Compared with LDACS the findings are only partially reusable since the pure FPGA-based implementation does not contain any mechanisms for transferring data between the software and hardware domains. Moreover, AAC uses very special but less computationally intense decoding mechanisms.

An interesting related project by Drozdenko et al. investigated the implementation of a wireless transceiver implementation using a hardware/software codesign approach [20]. The focus of this work was a high-level practicability proof of an 802.11a wireless internet access (WLAN) transceiver prototype based on reconfigurable hardware and software to create an adaptable platform for implementing future evolving standards. Taking into account tradeoffs between possible implementation variants, the researchers could identify room for optimization of their resulting system where real-time performance criteria were not met. The maximum achieved data throughput of the decoding system turned out to be not sufficient for their needs.

In summary, there is no system implementation reported in the literature that could be directly applied to LDACS. Implementation analyzes of individual blocks of the decoding chain do exist, but do not consider the overall performance of the system, while codesign approaches are usually very specific, or if generalized, underperforming. With a view to an effective implementation, this work focuses on an analysis of the individual decoding blocks, taking into account also the overall system performance. This system view guides the ultimate design partitioning.

IV. DESIGN CONSIDERATIONS

As mentioned in Section II-C, a pure software implementation of the LDACS encoding and decoding system, such as a SystemC model, does not meet the performance and latency requirements. It is more promising to evaluate if individual

blocks of the decoding chain in Fig. 3 can be favorably implemented in hardware, specifically in an FPGA module, to speed up the decoding process. Casting the entire decoding system in hardware, on the other hand, might have disadvantages such as a higher implementation and testing effort and inferior maintainability of the resulting system.

At the beginning of this analysis, the LDACS receiver must be considered as a whole, which consists not only of pure decoding. The receiving branch thereby includes an analog-to-digital converter, which receives a time-domain signal and forwards it to a FPGA-based signal preprocessing. There, in addition to filtering, a coarse synchronization of the data takes place, which are then converted into the frequency domain using a fast Fourier transformation (FFT). Subsequent fine synchronization, including channel estimation and OFDM frame demapping, already takes place in software, utilizing a direct memory access (DMA) mechanism to transport data from the FPGA to the software-based processing system. The following block-by-block analysis therefore assumes that the data to be decoded is already present in the software domain.

A. HELICAL DEINTERLEAVER

The data processing of the decoding starts with a helical deinterleaving, i.e., the incoming stream of data bits is permuted by writing the data helically bit by bit into an array and then reading it out line by line. As this step does not require intense computing effort and is only moderately memory intensive, the performance of hard and software implementation will not differ significantly. Implementing such system parts in an FPGA, however, may lead to increased implementation effort in terms of costs and time. Therefore, and also because the SystemC model did not show any problems regarding the processing time of the helical deinterleaver, it was decided to prefer the software form of this algorithm.

B. CONVOLUTIONAL DECODER

After deinterleaving, data need to be further processed reversing the convolutional encoding process by applying the Viterbi algorithm to the binary data stream. Based on the so-called trellis a maximum likelihood search defines the Viterbi algorithm output. The trellis is a shift register structure in which previous input bits of the algorithm, together with the current input bit, form the basis for deciding on the next output bit. This first error correction mechanism can handle a short series of errors followed by a longer period of error-free bits well, but fails for large error bursts [9]. Since the burst error form is problematic for the Viterbi algorithm, adding an interleaver in front of this mechanism is important. The helical deinterleaver used in this case rearranges the incoming data bits so that bursts are resolved and spread, which ensures a better error correction capability of the overall system.

In addition to the modulation method described in Section II, a convolutional coding rate is agreed between transmitter and receiver where three options are available for LDACS as 1/2, 2/3, and 3/4 coding. A rate of 1/2 means that a convolutional code is applied which converts each input bit into two output bits. Coding rates 2/3 and 3/4 use a similar output creation process. The difference here is to discard output bits, with for example two discarded output bits per 3 input b for the 3/4 encoding rate.

With respect to implementing this function block, the SystemC simulation and also the time it takes the algorithm to process a certain amount of data show a high degree of complexity. Since this part of the system, when implemented in software, causes most of the timing problems for real-time decoding, it is a clear candidate for an implementation as an FPGA module, combining the advantages of offloading it from the software processor and allowing a much shorter processing time.

C. BLOCK DEINTERLEAVER

One of the drawbacks of the Viterbi algorithm is that it is probability-based. It may not be able to correct all sorts of errors, and it is not able to give information on the correctness of the output data stream as it outputs just the most probable input to the convolutional encoder on the transmitting side. Therefore, further processing is necessary. The data are handed over to a block deinterleaver where after Viterbi decoding eight output bits are packed to one byte, from now on called a data symbol. The deinterleaver again performs data permutation, but on a symbol level, whereby the data symbols are sorted column by column in a 2-D array and passed to the following Reed–Solomon decoder row by row. This mechanism is rather memory intensive than characterized by high computational requirements. Therefore an implementation in software would be preferred.

D. REED–SOLOMON DECODER

The next step is Reed–Solomon decoding, which again is able to correct errors based on check symbols that have been added at the end of each data block in the encoding process. The block size before adding correction symbols varies from 66 to 206 B, with 8 to 22 correction/check symbols added. The check symbols are calculated in the transmitter and contain information of all previous data symbols within their block to be able to detect up to n erroneous symbols and correct up to $n/2$, with n being the number of check symbols.

The main advantage of this algorithm, besides the error correction capability, is the identification of finally erroneous data blocks, i.e., the information whether a data block is conclusively valid. Since the reverse computation of Reed–Solomon coding is known to be computationally expensive, this module is also a candidate for implementation in hardware. Additionally, the SystemC simulation showed probable problems in terms of processing time, which makes an implementation as an FPGA module indispensable.

E. DERANDOMIZER

Finally, data must be derandomized where data bits are fed to a linear feedback shift register combined with two XOR gates defining the derandomized and fully decoded data output. Owing to the short processing time of the function block in

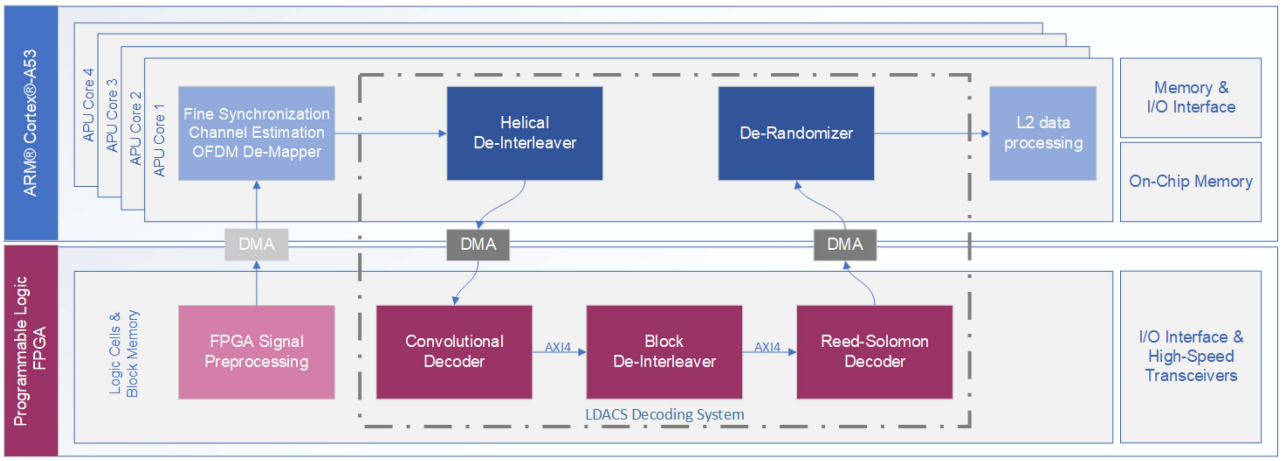

FIG. 4. **LDACS decoding chain architecture on the platform.**

software together with the previously mentioned advantages of software implementations in general, a software solution for the derandomizer is preferred.

F. INTERFACING BETWEEN HARD AND SOFTWARE BLOCKS

Analyzing the individual functional blocks alone is not sufficient for an efficient system design. In addition, also the data transport between the blocks must be considered, especially if it involves interfacing between software and hardware domain. Since LDACS signal processing will be based on an embedded platform with limited computing power, the impact on the overall system performance can be significant. This exchange is usually implemented in form of a direct memory access, whose processing time cannot be neglected. The total data transfer time consists of the actual data exchange, which can be considered very fast, and the setup of the DMA framework. Here, the largest latency components arise from allocating memory, setting up signal handlers and initializing the core transaction in the DMA driver.

The interfacing aspect is especially relevant for the design choice of the block deinterleaver between the Viterbi and Reed-S-olomon decoder. Seen from a pure functional viewpoint, this block would be preferably implemented as a software module. However, two additional data transfer blocks would be needed between hardware (Viterbi and Reed–Solomon decoder) and software (deinterleaver) domains. These domain changes would introduce additional, nonnegligible latencies. Therefore, the best choice from a system perspective is to implement the block deinterleaver also as an FPGA module.

The resulting hardware/software partitioning of the functional decoding blocks therefore starts with a helical deinterleaver in form of a software module, followed by the Viterbi decoder, block deinterleaver and Reed–Solomon decoder implemented as FPGA cores (Fig. 4). Finally, the data are derandomized, which again is implemented as a software module. This platform-independent analysis was applied to a specific implementation platform in the following to evaluate the conceived hardware/software codesign.

Based on this setup, analyzes could also be performed for the coding chain, since this implementation also influences the overall performance of the system. However, the coding scheme chosen in LDACS, such as convolutional and Reed–Solomon coding, requires much less effort on the encoding side. The potential advantage of speeding up the encoding blocks in hardware is therefore very limited and would be offset by the disadvantages already mentioned, such as low maintainability.

V. PERFORMANCE EVALUATION

When implementing a real-time decoding system, the main performance criteria are data throughput achieved and system latency, which define the usability with respect to the system requirements. In addition, a decoding system should also be able to correct errors that occur during the transmission of data from the transmitter to the receiver. For the implemented system, these metrics were tested to evaluate the performance of the system and validate its real-time capability for LDACS.

A. HARDWARE PLATFORM

As a platform for system implementation and evaluation, a hybrid of software processing unit and FPGA was chosen. The hardware used is a ZYNQ MpSoC from Xilinx, which is based on an ARM Cortex A53 quad-core processor system and an Ultrascale+ FPGA, combined in a single chip. The design goal for the software decoding part is to run on a single CPU core and use as few resources as possible, since higher layer functions will also be operated on this platform. Based on the previous design decisions, hardware blocks were implemented in the hardware description language VHDL, software parts were programmed in C/C++ running in a Linux based operating system. Theoretically, a bare-metal implementation of the software decoding blocks without an underlying operating system would be a possible variant. However, such implementations are also associated with many limitations. In a Linux environment, e.g., drivers for DMA mechanisms are available to ensure secure data transport. In addition, after decoding, the data from the decoding system are passed to

TABLE 1. FPGA Resource Utilization With the Occupied Resources on the Used FPGA in Percent

Decoding unit	CLB LUT as logic	CLB LUT as memory	CLB flip-flops
Viterbi decoder	3647 (1.70%)	26 (0.02%)	2409 (0.56%)
Block deinterleaver	152 (0.07%)	12 (0.01%)	211 (0.05%)
Reed–Solomon decoder	1476 (0.69%)	89 (0.06%)	1403 (0.33%)
DMA	2597 (1.21%)	227 (0.16%)	4665 (1.09%)
Overall utilization	7872 (3.67%)	354 (0.25%)	8688 (2.03%)

higher level applications that rely on an operating system environment, so the bare-metal variant would require additional cross-domain data transport and effort. These limitations combined with the low probability of speeding up the decoding tasks by doing so provided the basis to design the software parts of the system for a Linux-based environment.

The mentioned system-on-chip with necessary peripherals, such as power supply, network connection, etc., is included in the Xilinx ZYNQ Ultrascale+ MpSoC ZCU102 evaluation board. The ARM Cortex A53 processor thus runs an operating system based on an embedded version of Linux called Petalinux. The decoder blocks assigned to the hardware domain in Section IV were synthesized together with DMA blocks in the FPGA and form one of the two implementation domains together with their associated software drivers. The software decoding parts were built around this FPGA framework. Fig. 4 shows the final data flow including an overview of the system where each decoding block is shown in its associated system domain.

In detail, a clock rate of 200 MHz was used for the decoding blocks in the FPGA, with internal data transmission based on an AXI4-Lite IP interface [21]. This handshake-based data transfer was implemented with a maximum bit width of 64 b, with which, e.g., data are passed from the DMA to the decoding blocks. This results in a minimal latency of one clock cycle per data packet, allowing data to be transported internally at a rate of 1.6 GB/s. Between the decoding blocks, the data width is adapted to the internals of the coding blocks, i.e., the Viterbi decoder generates one output bit per two input bits at 1/2 coding rate, so that the bit width of the AXI4 interface also transmits only one bit per clock cycle at this point. This ensures the decoding with minimum internal delay. In terms of FPGA resources, Table 1 gives an overview of FPGA configurable logic block usage for each hardware decoder part comparing different lookup-table (LUT) forms and flip-flops. The FPGA resource consumption of the LUTs of the overall system including data preprocessing, coarse synchronization, etc., for the chosen FPGA currently amounts to 63%.

B. TEST SETUP

In order to test the implemented system design, a reference system is needed to verify the data correctness after the decoding process and the error correction capability of the system. For this purpose, a MATLAB system model was created, designed to export encoded data in form of raw data files, which then directly serve as input for the real system under test. As a first step, the layer 2 source, which in reality would be the data input to the layer 1 encoding system, was simulated in MATLAB in the form of a random data generator. These data packets were then encoded using built-in MATLAB functions according to the LDACS standard, with a raw data file saved before and after each encoding block. These files in turn serve as input to the system under test, which runs on the before mentioned ZYNQ Ultrascale+ MpSoC hardware platform. All decoding parts, for which a software-based implementation is more suitable, run on the ARM Cortex A53 processing system and all hardware-based parts were synthesized in the FPGA section of the system-on-chip, as shown in Fig. 4. This raw data-based test setup allows an encrypted data set to be passed to a specific decoding block and then its output file to be compared with the MATLAB data to verify correctness. Fig. 5 shows a schematic representation of the setup used for testing.

In a next step, intentional deviations were applied to the input data prior to decoding to simulate errors in various forms. The type of errors that occur is defined by the expected transmission channel. Commonly, an additive white Gaussian noise channel is considered, which can be characterized as a number of independent noise sources. The probability density function of the occurring errors can be described by a Gaussian curve. The resulting errors are randomly distributed over the input bit stream, with their probability depending on the signal-to-noise ratio [22]. Furthermore, other fault cases must be considered, where sources are assumed to block the channel or at least parts of the channel for a certain period of time. This leads to a larger section in the input bit stream of the decoding system where no bit was correctly transmitted due to the signal interference, represented in the form of burst errors.

With regard to the actual error correction, the Viterbi decoder and the Reed–Solomon decoder were examined more closely, as these two blocks are the only ones that have error correction capabilities. Other parts of the decoding system basically only reorder data, such that other decoding blocks can handle errors more effectively. The system under test for these two decoding modules was an adapted FPGA design with direct access to the individual coding blocks to get to know their decoding performance. As a first error correction block, the Viterbi decoder typically has different performance depending on the type of error [9]. Long burst errors are rather difficult for this algorithm to handle, whereas randomly distributed single errors at different positions are easier to detect and correct. The Reed–Solomon code, on the other hand, is more independent of the type of error, with its ability to correct generally depending on the number of errors within a Reed–Solomon codeword, not on their position or type.

Since the individual decoding blocks are combined to form a complete decoding system with interleaving and randomization mechanisms, the errors, which at first sight seem difficult to correct, turn out to be less critical. With burst errors at the input of the decoding system being reordered by the helical deinterleaver before passing the data to the Viterbi decoder, large bursts of erroneous data bits are broken up and evenly distributed over large interleaving blocks. The design of the

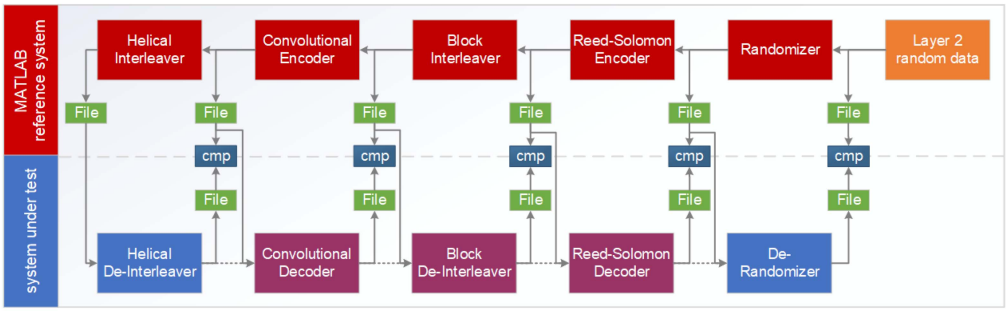

FIG. 5. LDACS test setup system under test– MATLAB reference system.

TABLE 2. Error Correction Ability of the Decoding System

Modulation	Coding rate	Block length [in bit]	Tolerable error abs[in bit]	percent
QPSK	1/2	9768	650	6.65 %
	2/3	9768	300	3.07 %
	3/4	9768	150	1.54 %
16QAM	1/2	19536	1400	7.17 %
	2/3	19536	600	3.07 %
64QAM	1/2	29304	1900	6.48 %
	2/3	29304	900	3.07 %
	3/4	29304	500	1.71 %

TABLE 3. Comparison of Processing Times of a Standard LDACS Data Block (QPSK 1/2) and One Super-Frame [in μs]

Decoding unit	SystemC	HW/SW codesign
Helical deinterleaver	227.44	227.44
Viterbi decoder	75,827.00	25.23
Block deinterleaver	6.89	6.10
Reed–Solomon decoder	214.12	9.10
Derandomizer	19.56	19.56
DMA framework	0.00	336.26
Standard LDACS data block	76,295.01	623.69
Super-frame overall	1,410,196	11,581

helical deinterleaver was chosen to allow the Viterbi decoder to operate optimally, resulting in higher overall decoding quality. In case of randomly distributed errors at the system input, there is a low probability that these errors will be rearranged in the helical deinterleaver to form datasets with high error density. This would lead to an increase in error density at the output of the Viterbi decoder as well. The following block deinterleaver distributes potential Viterbi output data blocks with high error density to multiple Reed–Solomon data blocks. This prevents individual Reed–Solomon codewords from being loaded with many errors, while there are completely error-free codewords before and after this section.

VI. RESULTS

A. ERROR CORRECTION

The results derived from the independent block analysis for the Viterbi decoder show a high dependency of the error correction performance on the error form as expected. An initial analysis examined burst errors of varying lengths, which may occur under the assumption that system input data were rearranged in the helix deinterleaver, creating contiguous error chains. As a first result, long error vectors of up to 40 b at the input enclosed by an otherwise error-free data stream led to a relatively small number of up to 7 consecutive residual errors at the output. These errors would in turn finally end up in a Reed–Solomon data block, where they could be easily corrected.

To further stress the Viterbi decoder, it was fed with recurrent error bursts with error-free sections of a specific length in between. This revealed a large dependency of the error correction potential on the length of the error-free section surrounding it, i.e., whether the Viterbi decoding block has time to return to a stable, error-free inner state. Already with burst errors of 5 b length an error-free section of 43 b is necessary to tolerate errors again without generating any errors at the output. Since this error form, as mentioned before, is extremely unlikely in the overall system due to the reordering in the helical deinterleaver, this stress test represents a worst-case scenario. For single randomly distributed errors, a similar overall picture emerged, i.e., for single errors, besides their rate, mainly the length of the error-free sections in between counts to ensure error-free decoding. In general, a single error enclosed by 10 error-free bits can be completely corrected, whereas even a slight reduction of the error-free section at the decoder output leads to significant output error density.

Conversely, the Reed–Solomon decoding algorithm has a fixed error correction capability by definition. This means that the decoder is able to detect up to n errors and correct up to $n/2$, where n is the number of check characters per Reed–Solomon block. Examining the implemented algorithm, these properties were confirmed by adding errors to data blocks, varying not only just in the number of errors but also in their position within a Reed–Solomon data block. Another advantage of this decoder is the additional information provided. The Reed–Solomon algorithm can be used to determine

TABLE 4. Data Throughput of the LDACS Decoding System

Modulation	Coding rate	Super-frame size [in bit]	Data rate [in Mb/s] required	Achieved HW/SW	Achieved SystemC	HW/SW Real-time factor	SystemC Real-time factor
QPSK	1/2	180472	0.75	15.58	0.13	20.72	0.17
	2/3	180472	0.75	15.16	0.13	20.16	0.17
	3/4	180472	0.75	14.95	0.13	19.88	0.17
16QAM	1/2	356296	1.48	20.82	0.13	14.02	0.09
	2/3	356296	1.48	20.09	0.13	13.53	0.09
64QAM	1/2	532120	2.22	23.53	0.13	10.61	0.06
	2/3	532120	2.22	22.53	0.13	10.16	0.06
	3/4	532120	2.22	22.06	0.13	9.95	0.06

whether the errors in the input data can be corrected, which makes it clear at the output of the Reed–Solomon decoder whether a dataset contains further errors or not. This information is especially important for further processing, as the reliability of the data can be rated.

For the overall decoding performance, the tests showed different outcomes for each possible combination of modulation and coding rate, with results summarized in Table 2. In a realistic worst-case scenario where the radio channel is disrupted during a certain time interval, burst errors occur at the input of the decoding system. These are picked apart by the helical deinterleaver and forwarded to the Viterbi decoder. This in turn generates at its output the most likely transmitted data sequence for the given input sequence, which is packed into blocks and passed on to the block deinterleaver. For data sections at the Viterbi decoder input that have a high error density and therefore also generate a high error density at its output, the block deinterleaver distributes erroneous data over multiple Reed–Solomon data blocks, thus increasing the resulting correction performance. Finally, the data are reordered via the derandomizer, which reconstructs the final decoded data stream in the best possible way. For a data block of 9768 b (standard length in LDACS with QPSK), Table 2 gives a error correction capability of 6.65 % error bits per block, with the higher coding rate in each case having a lower error tolerance in relation to the data throughput. This fact highlights the tradeoff between coding efficiency and error correction capability, since for a coding rate of 1/2, two bits must be transported for each actual bit of data, while for a coding rate of 3/4, this overhead decreases with 4 b transported for 3 bs of data.

Furthermore, Table 2 seems to suggest that the results for identical coding rates and different modulation methods are approximately the same. However, this impression is misleading, since with higher modulation rate the possibility for errors on the transmission channel also increases. With 64QAM, each frequency carrier within a symbol carries 6 b, while with QPSK it carries only 2 b. Considering each carrier as a complex value represented as a point in a 2-D complex plane, an error means that a point in this plane deviates from its position as a result of the transmission. While 64QAM modulation allows 64 points arranged in a regular grid in this 2-D space, there are only four possible points in the same 2-D area with QPSK. The received signal is then assigned to one of the possible transmitted points of the modulation used, simply choosing the closest point. In view of this fact, a received signal or its complex representation, respectively, can be interpreted correctly more easily for QPSK, since there is a greater error tolerance from the very beginning. With 64QAM, even small deviations cause the transmitted data to be misinterpreted as an erroneous value, since the respective data points in the complex plane have a very small distance. This leads to a generally higher error probability.

B. REAL-TIME PERFORMANCE

Considering the latency of the chosen implementation approach, it can be seen, as shown in Table 3, that the test system meets the requirements without any problems. The tested system has a decoding time of well below 1 ms per standard LDACS data block or less than 12 ms for a complete Super-Frame, so that the specification of 200 ms end-to-end latency is not affected. The chosen implementation therefore leaves enough flexibility for all other system parts contributing to the system latency to stay below the required maximum latency, so that LDACS can be used for digital speech. On the other hand, the SystemC implementation, which serves as a reference, would cause problems since the Viterbi decoder already requires more than 76 ms for a standard LDACS data block and thus the overall latency of 200 ms would not be met.

With respect to the real-time processing performance, the achieved data throughput represents the most important parameter, since it gives an indication about the usability of this implementation for real-time operation. Since standard frames in LDACS, so-called super-frames, have a fixed length of 240 ms a minimum required data rate can be specified depending on the selected modulation and coding rate. Table 4 shows a comparison of the required data rates for different LDACS coding parameters together with the size of the corresponding super-frames in bits. In addition, the achieved decoding data rates were analyzed together with the overall processing times and the time spent on data transport. The throughput of the SystemC simulation, as a potential representation for a software-only implementation, was compared to the chosen combined hardware/software solution.

The real-time factor presented in the table is calculated by dividing the achieved data rate by the required one, with values greater than 1 indicating faster decoding than needed.

The results show that a pure software implementation is by no means sufficient to meet the performance requirements.

In summary, the platform used has a high degree of over-performance, so that the use of a less powerful system for a pure decoding system would be obvious. Since the basic idea for the selected test platform was to use it not only for layer 1 processing but also for higher layer receiver parts, the selected system on chip (SoC) platform still appears to be a suitable choice.

A comparison with other LDACS implementations is difficult, as only a few prototypes are currently being developed. The associated implementation details, in particular those of the decoding chain, are not publicly available, which means that the performance of the system mentioned here cannot be compared with any other system as there is no reference.

VII. CONCLUSION AND FUTURE STEPS

In this article, we have reported about the first, to the best of our knowledge, working decoder implementation of the upcoming aeronautical communication standard LDACS. Specifically, we investigated design choices regarding the hardware/software partitioning that are crucial to achieve the performance goals. Our efforts were focused on the decoding chain because this is significantly more complex and resource consuming than encoding. Encoding can fully and conveniently be done in software, whereas especially the real-time trellis decoding is impossible without hardware support. Comparing the achieved and required data rate in Table 4, we see that 64QAM 3/4 shows 22.06 Mb/s achieved and 2.22 Mb/s required throughput, which is an over-achievement by a factor of about 10. In a detailed analysis, the software Viterbi decoder takes up to 99% of the processing time of the entire decoding process, which can be significantly reduced to about 10% by implementing this algorithm in an FPGA. An additional benefit of the FPGA implementation of the Viterbi decoder, block deinterleaver, and Reed–Solomon decoder is that it allows for parallelization and pipelining between these three blocks so that the output of one decoder is immediately passed to the next without waiting for the previous one to finish a block of data.

Together with the processing times of the individual decoding blocks, the analysis revealed that also the time required for data transfer between the blocks has a significant impact on the overall system performance. The initialization of the DMA mechanism and the time for data transport between hardware and software domains therefore cannot be neglected. Since the DMA setup takes a few hundred microseconds per transaction regardless of the size of the data block to be transported, the impact is higher for small data blocks than for larger ones. The decoding block analysis and resulting implementation show that LDACS can be implemented such that real-time requirements of the standard can be met.

In the future, the data transfer mechanisms between hardware and software domain should be improved in particular. This part of the system was not initially seen as a potential limiting factor, but it has shown a high impact on the overall performance. In particular, the number and size of individual DMA transactions could be further optimized to achieve even higher throughput, which would also lead to better scalability for future systems.

REFERENCES

[1] ICAO, *Annex 10 to the Convention on International Civil Aviation: Aeronautical Telecommunications, Volume I - Radio Navigation Aids*, 7th ed. Jul. 2018.

[2] *Eur. Commission-Directorate Gen. for Mobility and Transp.*, "SESAR 2020: Developing the next generation of European air traffic management," 2014. [Online]. Available: https://www.sesarju.eu/sites/default/files/-factsheet_SESAR-final-web.pdf

[3] ICAO, *Manual on VHF Digital Link (VDL) Mode*, Doc 9776, AN/970, 2nd ed., 2015.

[4] N. Neji, R. de Lacerda, A. Azoulay, T. Letertre, and O. Outtier, "Survey on the future aeronautical communication system and its development for continental communications," *IEEE Trans. Veh. Technol.*, vol. 62, no. 1, pp. 182–191, Jan. 2013.

[5] C. Rihacek et al., "L-band digital aeronautical communications system (LDACS) activities in SESAR2020," in *Proc. Integr. Commun. Navigation, Surveill. Conf*, 2018, pp. 1–8, doi: 10.1109/ICNSURV.2018.8384880.

[6] M. Schnell, U. Epple, D. Shutin, and N. Schneckenburger, "LDACS: Future aeronautical communications for air-traffic management," *IEEE Commun. Mag.*, vol. 52, no. 5, pp. 104–110, May 2014.

[7] S. Brandes, U. Epple, S. Gligorevic, M. Schnell, B. Haindl, and M. Sajatovic, "Physical layer specification of the L-band digital aeronautical communications system (L-DACS1)," in *Proc. Integrated Commun. Navigation Surveill. Conf.*, 2009, pp. 1–12.

[8] M. Mostafa, M. Bellido-Manganell, and T. Gräupl, "Feasibility of cell planning for the L-band digital aeronautical communications system under the constraint of secondary spectrum usage," *IEEE Trans. Veh. Technol.*, vol. 67, no. 10, pp. 9721–9733, Oct. 2018, doi: 10.1109/TVT.2018.2862829.

[9] I. Glover and P. Grant, *Digital Communications*. London, U.K.: Pearson Education, 2010.

[10] T. Gräupl, C. Rihacek, and B. Haindl, "LDACS A/G Specification SESAR2020 – PJ14-W2-60, European Union Horizon 2020 research and innovation programme," Edition 00.01.00, Dec. 2020.

[11] N. Zelkin and S. Henriksen, *L-band Digital Aeronautical Communications System Engineering–Concepts of Use, Systems Performance, Requirements, and Architectures*. Herndon, VA, USA, 2010.

[12] N. Schneckenburger, N. Franzen, S. Gligorevic, and M. Schnell, "L-band compatibility of LDACS1," in *Proc. 30th Digit. Avionics Syst. Conf.*, 2011, pp. 1–4.

[13] I. Swathi and S. Rajaram, "FPGA implementation of viterbi decoder for software defined radio applications," in *Proc. Int. Conf. Wireless Commun. Signal Process. Netw.*, 2017, pp. 2070–2073, doi: 10.1109/WiSPNET.2017.8300126.

[14] S. M. Dilek, B. Oers, and M. Kartal, "Reed–Solomon decoder hardware implementation for DVB-S receiver," in *Proc. 21st Signal Process. Commun. Appl. Conf.*, 2013, pp. 1–4, doi: 10.1109/SIU.2013.6531372.

[15] A. Abdallah, E. M. Feron, G. Hellestrand, P. Koopman, and M. Wolf, "Hardware/Software codesign of aerospace and automotive systems," *Proc. IEEE*, vol. 98, no. 4, pp. 584–602, 2010.

[16] K. Chih-Hung, C. Li-Chuan, F. Kuan-Wei, and L. Bin-Da, "Hardware/Software codesign of a low-cost rate control scheme for H.264/AVC," *IEEE Trans. Circuits Syst. Video Technol.*, vol. 20, no. 2, pp. 250–261, Feb. 2010.

[17] I. Bolsens, H. J. De Man, B. Lin, K. Van Rompaey, S. Vercauteren, and D. Verkest, "Hardware/software co-design of digital telecommunication systems," *Proc. IEEE*, vol. 85, no. 3, pp. 391–418, 1997.

[18] T. Wiangtong, P. Y. K. Cheung, and W. Luk, "Hardware/software codesign: A systematic approach targeting data-intensive applications," *IEEE Signal Process. Mag.*, vol. 22, no. 3, pp. 14–22, May 2005.

[19] M. S. Ibraheem, K. Hachicha, and O. Romain, "Fast and parallel AAC decoder architecture for a digital radio mondiale 30 receiver," *IEEE Access*, vol. 5, pp. 14638–14646, 2017.

[20] B. Drozdenko, M. Zimmermann, T. Dao, K. Chowdhury, and M. Leeser, "Hardware-software codesign of wireless transceivers on zynq heterogeneous systems," *IEEE Trans. Emerg. Topics Comput.*, vol. 6, no. 4, pp. 566–578, Oct.-Dec. 2018.
[21] ARM, "Amba axi and ace protocol specification." [Online]. Available: https://developer.arm.com/documentation/ihi0022/e/
[22] M. C. Liberatori, L. Coppolillo, L. J. Arnone, D. M. Petruzzi, J. C. Moreira, and P. G. Farrell, "Channel characteristics dependence of the performance of decoding algorithms for efficient error-control codes," in *Proc. XVII Workshop Inf. Process. Control*, 2017, pp. 1–6, doi: 10.23919/RPIC.2017.8211635.

S. KURZ received the B.Sc. degree in electrical engineering and the M.Sc. degree in embedded systems, from TU Wien, Vienna, Austria, in 2016 and 2019, respectively.

He is a hardware/software engineer in the field of aeronautical communications with a focus on the implementation of systems currently under standardization and research.

E. GRINGINGER received the MSc, MSocEcSc, and Ph.D. degrees with distinctions in computer science from TU Wien, Vienna, Austria.

He is Senior Lead Scientist with Frequentis with more than 10 years of experience in safety critical domains. He is an experienced research Project Manager. His research focuses on information management and data science.

Dr. Gringinger is a Member of EUROCAE working group an AI and works as technical reviewer for the EU.

C. RIHACEK received the M.Sc. degree in electrical engineering from TU Wien, Vienna Austria, in 1998.

He is a Senior Project Manager with Frequentis AG and has work experience as a Research Project Manager in several research projects.

Mr. Rihacek is a Member of the working group with ICAO, which works on the LDACS standardization.

T. SAUTER (Fellow, IEEE) received the Ph.D. degree in electrical engineering from TU Wien, Vienna, Austria, in 1999.

He is a Professor of automation technology with TU Wien and was the Founding Director of the center for integrated sensor systems, Danube University Krems, Wiener Neustadt, Austria.

Dr. Sauter is a Senior Administrative Committee Member of the IEEE Industrial Electronics Society.