electronics

MDPI

Article

Design and Development of a CCSDS 131.2-B Software-Defined Radio Receiver Based on Graphics Processing Unit Accelerators

Roberto Ciardi [1,2,*], Gianluca Giuffrida [2], Matteo Bertolucci [2] and Luca Fanucci [2,*]

[1] Department of Information Engineering, University of Pisa, 56122 Pisa, Italy
[2] Space Division, IngeniArs S.r.l., 56121 Pisa, Italy; gianluca.giuffrida@ingeniars.com (G.G.); matteo.bertolucci@ingeniars.com (M.B.)
* Correspondence: roberto.ciardi@phd.unipi.it (R.C.); luca.fanucci@unipi.it (L.F.)

Abstract: In recent years, the number of Earth Observation missions has been exponentially increasing. Satellites dedicated to these missions usually embark with payloads that produce large amount of data and that need to be transmitted towards ground stations, in time-limited windows. Moreover, the noisy nature of the link between satellites and ground stations makes it hard to achieve reliable communication. To address these problems, a standard for a flexible advanced coding and modulation scheme for high-rate telemetry applications has been defined by the Consultative Committee for Space Data Systems (CCSDS). The defined standard, referred to as CCSDS 131.2-B, makes use of Serially Concatenated Convolutional Codes (SCCC) based on 27 ModCods to optimize transmission quality. A limiting factor in the adoption of this standard is represented by the complexity and the cost of the hardware required for developing high-performance receivers. In the last decade, the performance of software has grown due to the advancement of general-purpose processing hardware, leading to the development of many high-performance software systems even in the telecommunication sector. These are commonly referred to as Software-Defined Radio (SDR), indicating a radio communication system in which components that are usually implemented in hardware, by means of FPGAs or ASICs, are instead implemented in software, offering many advantages such as flexibility, modularity, extensibility, cheaper maintenance, and cost saving. This paper proposes the development of an SDR based on NVIDIA Graphics Processing Units (GPU) for implementing the receiver end of the CCSDS 131.2-B standard. At first, a brief description of the CCSDS 131.2-B standard is given, focusing on the architecture of the transmitter and receiver sides. Then, the receiver architecture is shown, giving an overview of its functional blocks and of the implementation choices made to optimize the processing of the signal, especially for the SCCC Decoder. Finally, the performance of the system is analyzed in terms of data-rate and error correction and compared with other SW systems to highlight the achieved improvements. The presented system has been demonstrated to be a perfect solution for CCSDS 131.2-B-compliant device testing and for its use in science missions, providing a valid low-cost alternative with respect to the state-of-the-art HW receivers.

Keywords: CCSDS 131.2-B; software-defined radio; graphics processing unit; receiver; low-cost; earth observation; new space economy

Citation: Ciardi, R.; Giuffrida, G.; Bertolucci, M.; Fanucci, L. Design and Development of a CCSDS 131.2-B Software-Defined Radio Receiver Based on Graphics Processing Unit Accelerators. *Electronics* **2024**, *13*, 209. https://doi.org/10.3390/electronics13010209

Academic Editor: Duc Thanh Nguyen

Received: 29 November 2023
Revised: 21 December 2023
Accepted: 30 December 2023
Published: 2 January 2024

1. Introduction

In recent years, we have faced a process of commercialization of space exploration. New actors such as private companies, small and medium-size enterprises (SMEs), and start-ups are increasing their presence in what is called the New Space Economy (NewSpace). NewSpace is seeing enlarging participation of commercial companies in space exploration, with the purpose of developing faster, better and cheaper access to space and spaceflight technologies [1,2]. In this scope, the government agencies, which were the main agents in the past, now are not uniquely responsible for space missions. Big companies (e.g., SpaceX [3], Blue Origin [4], Virgin Galactic [5]) are investing their funds in researching

and developing new reusable launchers, hence making it increasingly affordable to launch products into space. At the same time, smaller companies are involved in the process, either by deploying and launching small satellites (e.g., CubeSats [6,7], SmallSats [8]), or by developing new, low-cost solutions dedicated to each part of a space mission (from ground segment to space segment). The effect that NewSpace is having on the field can be summarized as the democratization of space exploration: while the space agencies aim to uniform the space mission paradigm, for example by standardization, private companies address the research and development of innovative, commercial, low-cost systems for the aerospace industry. This process allows, on one side, to lower costs and speed up space mission deployment, and on the other side to promote the presence of even more agents in the sector [1].

In the past years, during the design of a space mission, each space subsystem was usually developed ad-hoc for the mission, with little if any re-use from other missions [9,10]. With the advent of NewSpace, the spacecraft industry is now evolving towards standardization to promote the re-use of technologies from previous missions [10,11]. Standardization has become a task of fundamental importance: indeed, the adoption of standard protocols has a large positive impact on the economics of a space mission by improving subsystem devices interoperability and promoting design reuse for subsequent missions. In this scope, the European Space Agency (ESA) is promoting the definition of a set of standards for communication, mainly through the Consultative Committee for Space Data Systems (CCSDS) [12] and the European Cooperation for Space Standardization (ECSS) [13]. Adhering to a standard, though, requires the verification of compliance with the standard itself. Single parts and components shall be tested to abide by the protocol and to intercommunicate correctly. At the same time, newer standards are required to be evaluated and assessed to be accepted and used in future missions. For this reason, Electrical Ground Support Equipments (EGSEs), hence instruments to be used on the ground to test the electronics to be launched, are of fundamental importance in the adoption of a standardized approach, allowing us to (1) validate spacecraft subsystems against the use of specific standards and (2) emulate the use of newer protocols to evaluate their characteristics and performance.

A communication standard defined in 2012 by the CCSDS for Telemetry application is the CCSDS 131.2-B. It is a flexible advanced coding and modulation scheme for high-rate telemetry, suitable for science missions and in particular for Earth Observation (EO) missions, where large amounts of data need to be transmitted from satellites to Earth, at high data rates. Being a recently defined protocol, it has currently never been deployed in any space mission. A limiting factor for the use of the CCSDS 131.2-B is represented by the complexity and high cost of the hardware systems required to be compliant with it and to reach the deserved performance. Both transmitters employed on satellites and receivers employed on ground stations are expensive to implement and require significant effort for the complexity of the standard itself. This factor is reflected also in the related EGSEs, with a consequent slowdown of the whole process of testing and using the protocol. Specifically, for what concerns CCSDS 131.2-B-compliant receivers, the state-of-the-art solution is represented by the Cortex High Data-Rate Receiver (Cortex HDR) [14]. It is a complex system developed by SAFRAN, which may be used in ground stations for receiving transmissions compliant with the protocol, but also in test facilities to test transmitters' capabilities before satellite launch. Although the Cortex HDR is a complete solution for employing the CCSDS standard, its high cost strongly limits its deployment. Indeed, as with other HW-based solutions, the Cortex HDR can cost hundreds of thousands of dollars, making its procurement unfeasible for most of the private companies and for agencies as well. In fact, not all the ground stations can be equipped with not one but many of these receivers, for multiple missions. Nonetheless, the high cost also limits the use of the receiver for testing standard-compliant transmitters. In the ESA roadmap for EO, the CCSDS 131.2-B standard has been selected to be the reference standard for future missions. For this reason, it is necessary to dispose of EGSEs compliant with this standard, for testing on-board and on-ground systems capable of adhering to the CCSDS 131.2-B. Also, it is

crucial to provide low-cost systems to allow a wide adoption of the standard, opening its use to different subjects in the scope of the New Space Economy.

With this in mind, this paper presents the work carried out for developing a low-cost CCSDS 131.2-B-compliant receiver based on a software implementation. This manuscript aims to extend the preliminary results presented in [15], where an exploratory implementation of the decoder part of the receiver, the object of this paper, was presented first with a discussion of its speed performance. In this paper, a detailed description of the whole architecture of the system will be given, focusing on its performance, both from error correction and decoding speed points of view. The presented receiver exploits the use of graphics processing units (GPUs) for developing a software-defined radio (SDR) solution capable of reaching high performance while maintaining a high level of flexibility with a much lower cost with respect to the state-of-the-art solutions. This system would allow for the easy and low-cost testing of CCSDS 131.2-B compliant systems, promoting the adoption of this standard for future mission. Also, for missions where requirements are relaxed, as science missions, a SW receiver would be an easy-to-deploy alternative to the state-of-the-art receivers.

The remainder of this paper is organized as follows:

- Section 2 describes the CCSDS 131.2-B protocol, focusing on the reason for its definition and the architecture of compliant transmitter and receiver systems;
- Section 3 delves into the details of a receiver system compliant with the standard, describing the rationale for implementing an SW-based solution and the development of its main parts;
- Section 4 shows the performance of the system in terms of both error correction and data rate, compared with the state of the art of CCSDS 131.2-B receivers;
- Section 5 draws the conclusion of the work and outlines future developments.

2. CCSDS 131.2-B Protocol for High-Rate Telemetry

The CCSDS 131.2-B standard was defined by the CCSDS in 2012 as a flexible advanced coding and modulation scheme for high-rate telemetry [16]. The standard has been defined to address the major challenges encountered while operating a transmission above 10 GHz, in particular at the Ka-band (25.5 GHz to 27 GHz). It is specifically intended for low-Earth orbit (LEO) satellites, which are visible from Earth in small time windows ($\sim$10 min), in which all data must be transmitted to the ground stations. In fact, the standard is applied to create the transmission signal on-board satellite by encoding and modulating the generated data, to improve the quality of the transmission over the noisy satellite–ground station channel in a small amount of time. There are different ways of using the Modulation and Coding (ModCod) techniques on spacecrafts, of which two are usable with CCSDS:

- Variable Coding and Modulation (VCM), where satellite elevation or time coordinates are taken into account to change the ModCod to maximize link efficiency;
- Adaptive Coding and Modulation (ACM), where both the satellite's position and environmental conditions are considered to maximize link occupancy.

For signal coding, many current LEO satellites make use of RS encoders, along with trellis codes. For example, the satellites of the Sentinel program [17] use 4D-8PSK Trellis Coded Modulation (4D-8PSK-TCM), which exploits RS coding. CCSDS 131.2-B instead is based on a turbo-like coding and modulation scheme based on Serially Concatenated Convolutional Codes (SCCC). This scheme makes use of a set of different modulation techniques (e.g., QPSK, 8PSK, 16APSK, 32APSK, and 64APSK) and a wide range of coding rates [16]. The large variety of modulation schemes available, together with a properly selected coding rate, allows the overall system to efficiently handle the available bandwidth, adapting itself to the variable conditions of the link. Specifically, the CCSDS 131.2-B makes use of link adaption strategies based on 27 different ModCods, applied based on VCM or ACM to overcome major challenges, such as satellite visibility (i.e., due to the satellite orbit), channel impairments (i.e., due to the wireless channel), or to guarantee specific

performance (i.e., in terms of data rate). Using different ModCods aims to maximize the transmission of data while dealing with varying channel conditions, where a worst-case sizing of the transmission parameters would result in unacceptable inefficiency, given the large dynamic of the possible attenuation during a satellite pass over the ground station.

A brief description of the transmitter and receiver sides of CCSDS 131.2-B based communication is now given.

2.1. Transmitter Side

The definition of the CCSDS 131.2-B is based on the application of efficient coding schemes that address the problem of bandwidth limitations in the timing-varying Ka-band channel. All CCSDS 131.2-B 27 ModCods employ efficient, high-speed data transmission schemes based on an SCCC derived from the Modem for High Order Modulation Schemes (MHOMS) project [18].

A transmitter compliant with the CCSDS 131.2-B standard executes different functions: error-control coding (based on serially concatenated convolutional coding), frame validation, transfer frame synchronization, physical layer (PL) framing, bit synchronization, and pseudo-randomization. The architecture for an encoder and modulator compliant with the CCSDS 131.2-B is described in the standard [16] and is shown in Figure 1. The figure identifies the functional blocks of the system and shows the logical relationships among them.

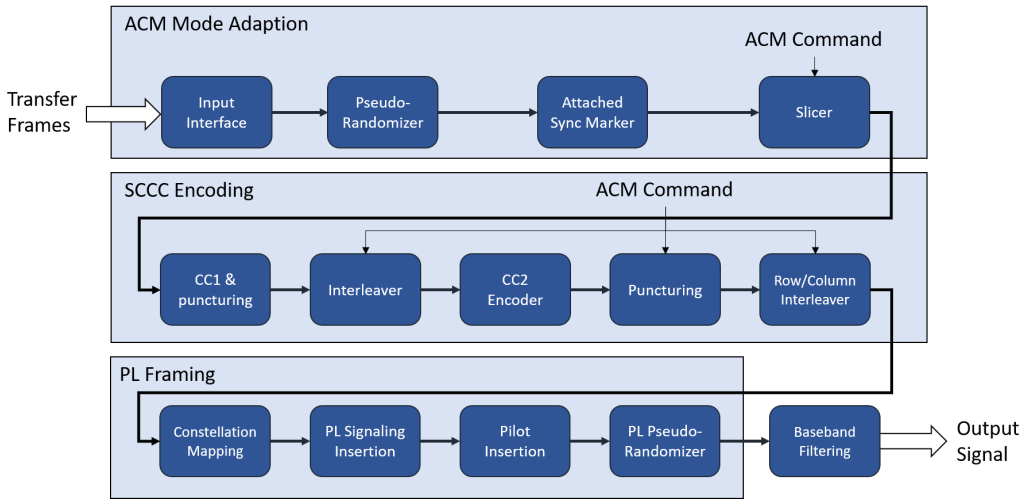

Figure 1. Transmitter Architecture.

The transmitting system accepts, as inputs, Transfer Frames (TFs) of fixed length, performs functions selected for the mission, and transmits a continuous and contiguous stream of physical channel symbols, obtained from PL frames. In particular, the SCCC encoder can be considered the main part of the system. It applies convolutional encoding employing two consecutive convolutional encoders (CC1 and CC2). Moreover, it applies the *puncturing* technique, which removes bits after the encoding following specific patterns, and the *interleaving* technique, which interleaves the data bits. The encoded bits are framed in a PL frame that is finally given as input to a baseband filter, applying a Squared-Root Raised Cosine (SRRC filter, defined in the standard). The input transfer frames have a specific format defined in the standard, which, together with the structure of the obtained PL frames, is crucial for supporting data-aided operations carried out at the receiving side.

The SCCC encoder of the transmitter encodes the so-called information blocks, composing output blocks with a dimension that is related to the used ModCod. The same ModCod scheme is applied to every 16 consecutive blocks, referred to as Codewords (CWs), that will be grouped to form a PL frame. The length of encoded symbol blocks after encoding and mapping to modulation symbols is constant (8100 symbols), regardless of the applied ModCod scheme; this facilitates frame synchronization at the PL for the receiver

side. Also, a set of predefined pilot symbols can be inserted inside each CW. Each CW is therefore divided into 15 subsections, each composed of 540 symbols and each followed by 16 pilot symbols. Overall, a PL frame is composed of a 320-symbol PL header, of which the first 256 symbols represent the Frame Marker (FM), and the remaining 64 symbols represent the Frame Descriptor (FD) and 16 $\times$ 8100-symbol CWs, eventually interleaved with 240 pilot symbols per CW.

The well-defined structure of the PL frame is crucial at the receiving side for synchronization. In particular, the FM is used to detect the start of a PL frame, as defined in the standard. The FD instead carries the information about the ACM format of the PL frame and the presence/absence of pilot symbols. These symbols are used on the receiving side for phase synchronization.

2.2. Receiver Side

Transmitter functions of the standard have to be implemented on satellites, for transmitting data towards Earth. On the other side, ground stations need to implement a standard-compliant receiver to receive and decode the signal. The general architecture of a receiver compliant with the CCSDS 131.2-B standard is detailed in Figure 2.

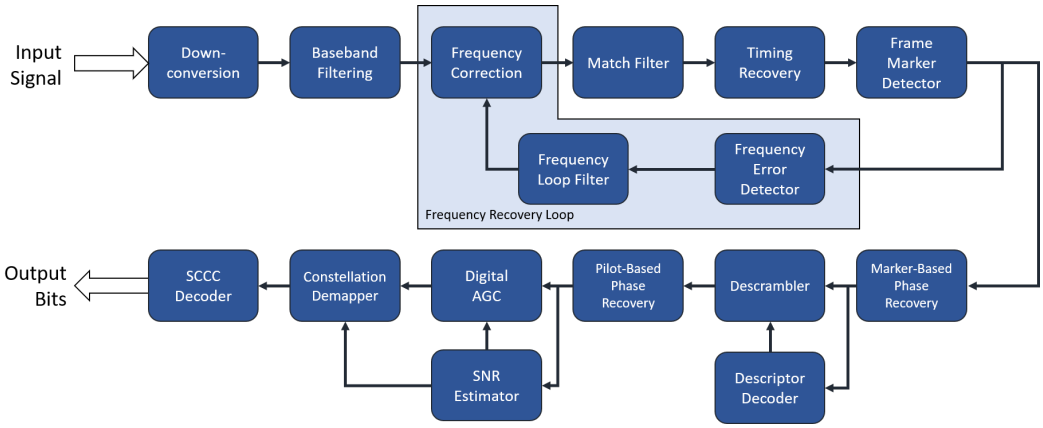

Figure 2. Receiver Architecture.

The receiver side is composed of a sequence of blocks that aims to extract and decode the information sent by the transmitting side. First, an ADC block should sample the incoming IF signal that is then down-converted to the base-band. This can be carried out for example by using a local digital quadrature oscillator. The signal is then filtered and decimated up to an integer number of samples required by the timing synchronization algorithm. The latter is used to recover the transmission clock; thus, it samples the data stream in the correct instant, to recover the amplitude and phase-modulated symbols. After that, the frame marker detector block allows the recovery of the PL frame structure by applying a correlation between the reference FM and the received bits. This is crucial to enhance the data-aided algorithms used for frequency correction and phase correction, whereas the previous blocks (i.e., timing recovery) must be capable of performing their functions without knowing the structure of the frame and in the presence of a frequency error. After compensating for the frequency impairment, the symbols are unscrambled according to the standard. This allows us to mitigate the effect of transmitter pseudo-randomization, re-establishing pilot values. At the same time, the descriptor decoder, analyses the FD of the frame, identifying the presence/absence of pilots and the ModCod used. The last part of the architecture is preceded by the estimation of the signal-to-noise ratio (SNR) and the correction of the amplitude of the symbols to match the reference constellation. This step is needed to compute the log–likelihood ratio (LLR) of the received bits by the constellation demapper module. Finally, the SCCC decoder uses this soft-bit information (i.e., the LLRs) to recover errors in the satellite transmission and produce the information bits.

The structure of the described receiver is quite complex, both for impairment recovery and for the internal architecture of the SCCC decoder. For decoding information at a high data rate, it is necessary to compute many operations on a large quantity of data in a small amount of time. For this reason, the state-of-the-art receivers for CCSDS 131.2-B communication are based on complex and expensive HW solutions, such as the aforementioned Cortex HDR [14]. The limiting factor of this receiver, and other HW receivers as well, is not only represented by the complexity of the design, but also by its very high cost, which restricts the testing and adoption of the standard. This is currently one of the main factors limiting the use of the CCSDS 131.2-B standard, considering also that it could be involved also for science missions where the performance requirements can be relaxed; hence, there is no need to reach the highest performance provided by a HW receiver (i.e., 500 Mbaud). For these reasons, this paper describes the implementation of a SW receiver to provide a low-cost alternative to HW receivers for promoting the use of the CCSDS 131.2-B standard by implementing easier and lower-cost EGSEs to test the transmission and satisfy the requirements of science missions. At the state of the art, as far as the author knows, the only SW solution for CCSDS 131.2-B is represented by the so-called SW EGSE [19]. It is a software simulator based on Matlab SW [20], capable of emulating the downlink communication compliant with the standard. This has been implemented for testing the communication and above all for testing the standard itself, as well as for assessing its capabilities. For its very low performance, it is impossible to use the SW EGSE both for testing real transmitters and for receiving data on ground stations. In particular, the SW EGSE is capable of decoding a single Codeword (8100 symbols) in, at minimum, more than 3 s, making its use unfeasible for any real-use case (in comparison, a HW receiver as the Cortex can decode about 60 k Codewords per second). The poor performance of the SW EGSE is due to the implementation of the EGSE itself, which was not explicitly developed for reaching high performance, but also due to the implicit limitation of SW implementation on Matlab.

In recent years, the evolution of general-purpose processing HW has increased software performance. Applications like synthesizers, modulators/demodulators, and encoders/decoders can now be implemented in software thanks to the use of ad hoc embedded systems or high-performance processors. This is the case for the so-called Software-Defined Radios (SDRs), which represent radio communication systems that exploit SW to implement components that are usually implemented on FPGAs or ASICs, such as filters, amplifiers, modulators, and demodulators. This approach has gained traction also in the space telecommunication sector [21–24]. A significant help to SDR performances has been given in recent years by General-Purpose Graphics Processing Units (GP-GPUs). In this scope, we have worked on the design and development of an SDR receiver, exploiting GP-GPUs to implement a system compliant with the CCSDS 131.2-B standard. The SDR approach offers manifold advantages with respect to the HW approach such as extensibility, flexibility, cheaper development and maintenance, upgradability, and potential future development [21,25].

The next section describes the development of the SW-based receiver, focusing on the implementation choices carried out to obtain good performance in terms of data-rate and error correction. Such performances are discussed and evaluated in Section 4.

3. Software Receiver Implementation

To develop a high-performance SW receiver, the parallelization of operations is a crucial task, carried out thanks to the use of GP-GPUs. GP-GPUs disperse the execution of a single instruction over all available data, boosting the application throughput, in accordance with the Single Instruction Multiple Data (SIMD) paradigm, on the other side's central processing units (CPUs) to follow the Single Instruction Single Data (SISD) paradigm, serially processing single data blocks. Large-scale data processing challenges are particularly well-suited for GP-GPUs, which provide the benefits of high-performance parallel processing at a cheaper cost than HW solutions like FPGAs. An example of an SDR

receiver exploiting GPUs is given in [26]. Naturally, all communication systems that need to handle massive volumes of data, like the CCSDS 131.2-B standard, can benefit from this concept.

Even though current CPUs are manufactured with such high frequency, they are often built to operate in a serial fashion, with the goal of minimizing latency for individual tasks. For this reason, CPUs are commonly not used for parallel processing, involving a large quantity of data. GP-GPUs, on the other hand, provide a versatile throughput-oriented processing architecture to easily support parallel computing. The main tasks covered by GPUs are related to graphics (e.g., computer vision, video rendering) or physics simulation. In general, the types of applications that must be run and the types of data that must be processed have a significant impact on the benefits and drawbacks of utilizing GPUs other than CPUs.

A comparison between the high-level architecture of CPUs and GPUs is depicted in Figure 3.

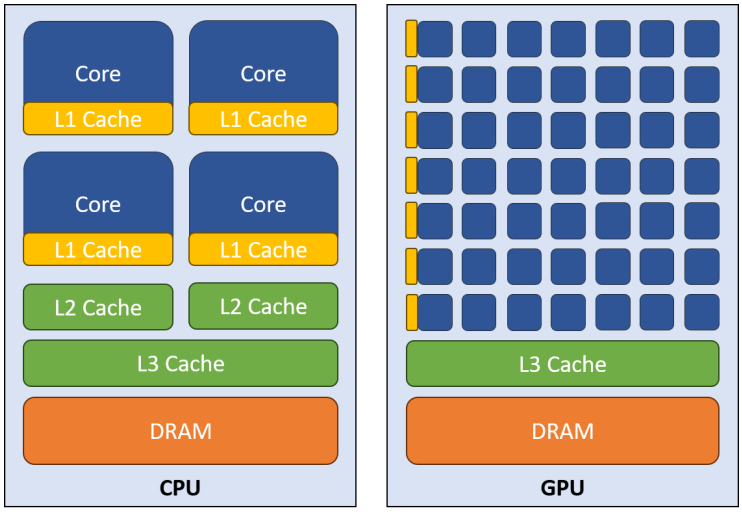

Figure 3. General CPU architecture (**left**) compared with general GPU architecture (**right**).

CPUs (Figure 3 left) are usually composed of a few cores (blue blocks), generally from 2 to 16, which are responsible for the parallel (or serial) execution of processes and threads. Typically, every CPU core carries out distinct tasks on distinct sets of data, with the aim of reducing the time lag between receiving an instruction and completing it. Additionally, each core handles requests of a higher priority that may arise from various peripherals, such as interrupts, exceptions, and user commands. With respect to GPUs, CPUs provide the best performance when:

- Task parallelism is necessary, meaning that several different jobs must be completed simultaneously while utilizing separate or even the same set of data;
- Tasks follow distinct instructions, meaning that each task's instructions provide its own objective;
- Each process/thread is programmed individually.

GPUs (Figure 3 right), on the other hand, are made up of thousands of cores (blue blocks), each of which is capable of processing thousands of threads at once, while concentrating on maximizing the application data throughput. Specifically, GPU threads perform the same instruction on distinct data blocks, exploiting the SIMD paradigm. With this method, enormous volumes of data can be managed effectively; all threads execute the identical instructions concurrently, but on distinct batches or segments of data. Also, GPUs usually are equipped with on-chip memories, which are crucial for allowing fast and eventually shared access to the data, by the threads. An efficient use of GPU memory, as well as of the several threads that can be instantiated, is of fundamental importance to guarantee fast access and processing of data.

In general, GPUs provide the best performance when:

- Data parallelism is necessary, meaning that the same activity must be executed on multiple data;
- A high-rate throughput is necessary.

This is needed for example for image processing, video rendering, and signal data processing when it is necessary to use algorithms to quickly run the same code on enormous amounts of data.

To develop a real-time SDR system compatible with the CCSDS-131.2-B standard, it was necessary to select an appropriate general-purpose GPU processor. Specifically, the NVIDIA Quadro RTX 4000 GPU, released by NVIDIA at the end of 2018 [27], has been selected. It has been programmed by means of the Computed Unified Device Architecture (CUDA), a C++ based framework, which allows us to develop functions to be executed in parallel on the multiple GPU cores. This NVIDIA GPU comprises 8 GB of memory and is equipped with 2304 CUDA cores, with a clock speed of 1005 MHz. Thanks to CUDA, the GPU is capable of instantiating 1024 threads per block and 65,535 $\times$ 65,535 blocks per grid, allowing us to parallelize the thread executions on the multiple CUDA cores. The adopted GPU is not the highest-performance GPU available on the market, but it has been chosen to demonstrate the potential of the system implemented on a low-cost device. Indeed, at the time of writing, the cost of this device is lower than one thousand $. The implemented Receiver has been developed on a workstation mounting a commercial CPU, an Intel Core i9-12900K.

To implement a flexible and easily maintainable SDR, each block of the Receiver has been developed following a programming paradigm called Factory Method Design Pattern [28]. This is an Object-Oriented Programming (OOP) technique based on the definition of abstract classes that are extended by sub-classes, which determine the nature of the instantiated object. In this way, the receiver blocks have initially been defined as abstract classes, with their specific communication interface, while the implementation is demanded to the programmer, with the possibility of easily employing and testing different solutions for each block. Following this design rule offers multiple benefits:

- It allows us to change the internal implementation of a block without modifying its interface, i.e., for adopting different algorithms;
- It ensures the possibility of changing each block modifying only a single module of it, without interfering with its entire functionality i.e., without affecting modules inside it or interfacing with it;
- It allows us to enable/disable the hardware acceleration support, creating both serial and parallelized code versions, for debugging and performance purposes;
- It compiles only a portion of the entire system, encapsulating possible errors inside the newly developed module and reducing the management time due to bug fixes;
- It allows for the extension of the developed blocks for eventually implementing new communication protocols, exploiting the same logical structure.

Thanks to this approach, the receiver has been implemented to be extremely flexible and modular, allowing alternating parallel code, executed on GPU, to serial code, executed on CPU. At the end of the day, the receiver was implemented as a class object, instantiating each single submodule and exploiting the Factory Method Design Pattern.

The architecture of the receiver, shown in Figure 2, has been logically divided into two main parts: the digital front-end section and the back-end section. The first one contains all the processing blocks that are responsible for recovering the signal impairments (i.e., from *Frequency Correction* block to *Digital AGC* block), whereas the back-end section only includes the generation and decoding of the soft-bit information (i.e., the *Constellation Demapper* and the *SCCC Decoder* blocks).

3.1. Front-End

Figure 4 describes the architecture of the implemented front-end section.

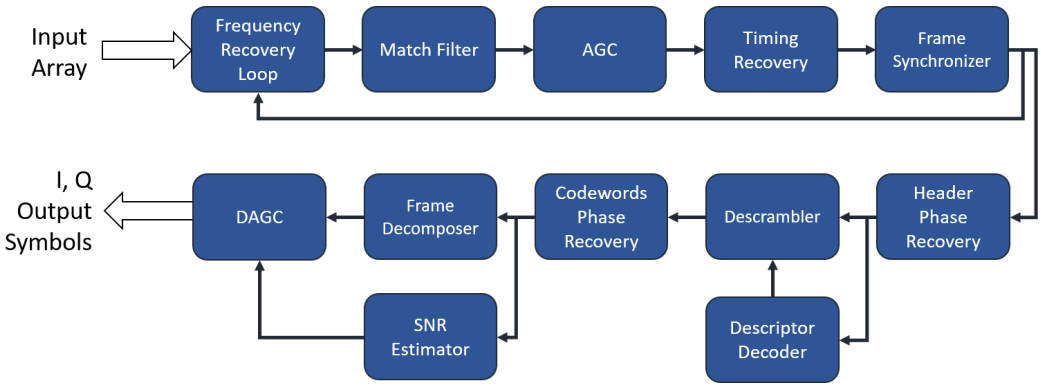

Figure 4. Implemented Front-End Architecture.

The organization of these modules slightly differs from the general architecture given in Figure 2, to be adapted to the SW implementation. The front-end modules need to be executed in a chain, where the result of each module is given to the subsequent module. For this reason, the parallelism of the front end is constrained, and it is necessary to optimize each single module to speed up this part of the receiver. Nonetheless, the front-end part of the system is the less expensive one in terms of processing power; hence, the GPU parallelism has been limited to the data parallelism of each single block. The studies conducted on the performance of the overall system have identified in the back end the bottle-neck of the system, as will be made clear in the next Section. For this reason, the implementation of the single blocks of the front end part of the system will not be discussed in this scope. Overall, to develop the front end part of the receiver, a hybrid paradigm has been used, exploiting the GPU when working on large amounts of data in parallel (i.e., when working on the whole PL frame) and exploiting the CPU when higher frequency is needed and less data are processed (i.e., when working on frame header and descriptor).

3.2. Back-End

The back-end subsystem is composed of the constellation demapper and the SCCC Decoder. The two blocks are responsible for transforming the received symbols in the output bits, reducing the errors due to channel impairments. As seen in the previous section, the front end of the receiver was not highly computationally intensive. On the other hand, the back end has been computed to be the most computationally intensive part of the receiver. For its recursivity and because of the several executed operations, the back end has been demonstrated to be the bottleneck of the receiver. For this reason, the development of the receiver has focused on the implementation of the back-end blocks on GPU, for optimizing their execution and improving the overall performance of the system. It is worth mentioning that both the constellation demapper and the SCCC decoder have been implemented entirely on the GPU with the input data that are moved to GPU memory before entering the back end and are moved back to the CPU (to be processed at the output) only at the end of the chain.

Specifically, the demapper block aims to transform the constellation symbols, corrected by the demodulator blocks, into soft bits represented as LLRs. Then, the SCCC decoder block eventually corrects transmission bit errors, providing the correct information bits. Figure 5 shows the architecture of the implemented back end, with a focus on the internal architecture of the SCCC decoder.

A detailed description of the development of the two back-end blocks (i.e., demapper and decoder) is reported in [15]. Here, an overview of the implementation choices is given to introduce the performance evaluation carried out in the next section.

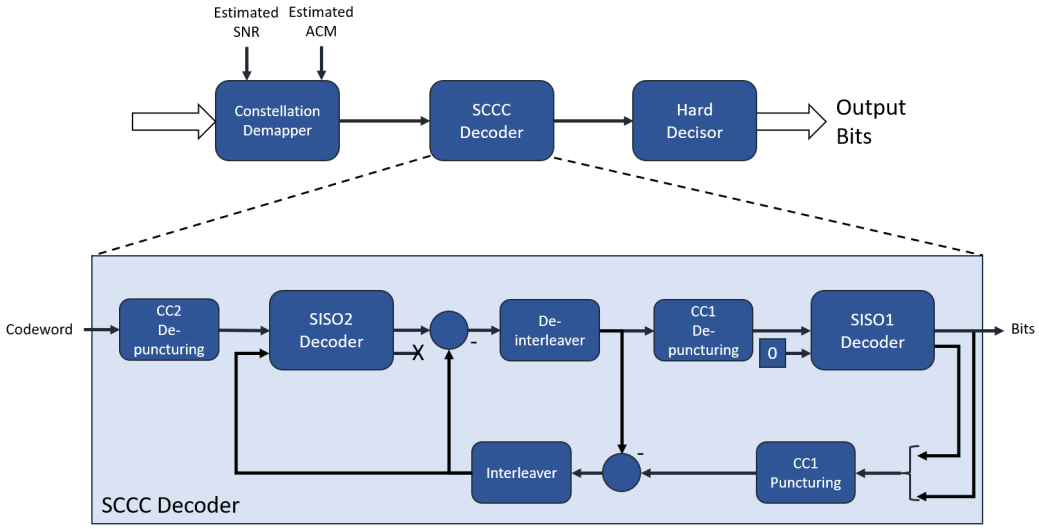

Figure 5. Implemented Back−End Architecture.

3.2.1. Constellation Demapper

For the implementation of the constellation demapper of the receiver, soft demapping has been considered due to its known superior performance with respect to hard demapping [29]. The input of the demapper module is the set of symbols of a PL frame (8100 symbols $\times$ 16 codewords), previously detected at front end and provided as a set of couples of in-phase (I) and quadrature (Q) elements. Also, the demapper gets as input the ModCod and SNR estimated at the front end, and processes each of the 16 parts of the PL frame (8100 symbols) to produce 16 Codewords to be given as input to the SCCC decoder.

The Demapper computes n Euclidean distances between each I and Q couple and the constellation points related to the used modulation, i.e., QPSK (ACM 1-6), 8PSK (ACM 7-12), 16APSK (AMC 13-17), 32APSK (ACM 18-22), 64APSK (ACM 23-27).

For instance, for a QPSK, there are four points, so the demapper would compute four Euclidean distances between each input point, with coordinates I and Q, and the four points of the constellation. Each computed distance is then multiplied by the estimated SNR.

Considering that a CW contains 8100 symbols, for ACM 27, the demapper has to compute more than 3 million (384 $\times$ 8100) distances and multiplications. These would be calculated in a serial process by using two nested iterative cycles, with a subsequent high computational cost. Instead, on a GPU, a single thread can compute a single Euclidean distance, with a number of threads executed in parallel that is equal to the number of distances to be computed. In this way, all the computations can be resolved in just one instruction, eliminating the iterative execution. After the computation of the distances, the demapper finds a set of m LLR values for each symbol, with m that is the modulation cardinality, proportional to the ACM. To do so, it computes the Jacobian logarithm [30], referred to as max* (1).

$$max^\star(a,b) \stackrel{\Delta}{=} ln(e^a + e^b) = max(a,b) + ln(1 + e^{-|a-b|}) \quad (1)$$

Specifically, the max* is computed on m sub-sets of distances, reducing the initial values to the final m values, for each I, Q couple. At the end of the demapper, 16 codewords with a size equal to $m \times 8100$ will be produced.

Thanks to the use of GPU, the Euclidean distances computation is processed in a few microseconds, while the max* function, because of its nature (i.e., the efficiency of the logarithm and exponential functions) and because of the large number of max* operations to be computed, represents the most computation-intensive part of the demapper.

The design of the demapper itself shows how an efficient use of GPU parallelization can increase the performance of the implemented receiver. These design choices will be further investigated in Section 4, where the performance of the receiver will be evaluated in comparison with the state of the art.

3.2.2. SCCC Decoder

The SCCC decoder can be considered the main section of the receiver, responsible for recovering the information bits from the encoded ones. Its input is the quantized soft bits from the constellation demapper module, representing the confidence of the encoded bits being one or zero. The information bits are decoded using a turbo-like structure as in Figure 5, where multiple iterations are used to improve the confidence of the bits, before the final hard decision. Also, for the internal architecture of the SCCC decoder, the Factory Method Design Pattern has been exploited, developing each single module to be highly flexible and easy to interface with the other modules.

The SCCC decoder implemented is based on the BCJR algorithm, defined by Bahl, Cocke, Jelinek, and Raviv [31]. It is an iterative Maximum A Posteriori (MAP) Trellis-based algorithm that aims to reconstruct the a posteriori probabilities by means of Soft-Input Soft-Output (SISO) decoders [32]. The nature of the SISO decoders, and the iterations needed to decode the signal, make the SCCC decoder the most computationally intensive part of the receiver.

Puncturing/de-puncturing and interleaving/de-interleaving modules aim to respectively remove/add redundancy bits and interleave/de-interleave bits according to a set of predefined rules. These modules have been developed following GPU-aided parallelism and they can be executed in a few microseconds.

The most important modules of the SCCC decoders are obviously the SISO decoders. Further details about the implemented BCJR algorithm can be found in [33]. Specifically, the trellis processing is performed in two steps: forward and backward. For this reason, the BCJR is also called the *forward-backward* algorithm. These steps must be performed iteratively, limiting the use of parallelization, and slowing down the whole process. Also, each step requires the computation of the already-mentioned $\max^\star$ equation, which is known to be computationally intensive. Overall, three operations are the most complex to be designed and accelerated for the SISO decoders:

- The forward recursion, which calculates the α values iteratively, limits the application of the SIMD technique because each value is dependent on the elements computed before it.
- The backward recursion, which calculates the β values iteratively, limits the application of the SIMD technique because each value is dependent on the elements computed before it.
- The $\max^\star$ that requires the computation of a logarithm and an exponential, both intensive functions due to the software abstraction needed.

To increase the performance-exploiting GPU programming as far as possible, different techniques have been used. First of all, the GPU memory has been exploited to guarantee fast and shared access to data for each thread. Then, to parallelize α and β computation, the windowing technique has been employed [32]. This procedure involves parallelizing the computation of sets of samples by computing the metrics in successive windows. With an effective number of windows, this procedure parallelizes the whole computation, without a significant implementation loss. The windows number is strongly related to the number of inputs of the SISO decoders, which is proportional to the used ModCod. The larger the number of windows, the larger the speed of the process. In our implementation, the ideal number of windows for ACM 1 has been computed to be about 15 windows, while for ACM 27, we could employ up to 60 windows, without significant loss in error correction performance. On the other hand, to avoid the slowing factor introduce by the Jacobian logarithm (1), we have chosen an approximation of it. Several approximations of this function have been proposed in literature [34,35]. One of these proposes to use the classic

max instead of the max* function, introducing an implementation loss in error decoding performance [32].

These choices are investigated in the next section, where the overall speed of the system is analyzed when using windowing and the simplified Jacobian logarithm, focusing on the introduced performance improvements and implementation loss.

4. SDR Receiver Performance Evaluation

The performance of the implemented system has been evaluated from two points of view, considering data-rate and error correction performance. To evaluate the system, a compliant SW transmitter has been developed to produce input data for the receiver. To decouple transmitter and receiver speed, the encoded bits were computed offline and then provided to the receiver as sets of thousands of PL frames. Also, for evaluating the error correction performance of the system, the channel was emulated by adding Additive White Gaussian Noise (AWGN) to the input. This has been used to compute the Bit Error Rate (BER) of the system, compared with the ideal BER given by the standard [36]. In this case, the input of the system was computed as a well-known Pseudo-random Noise sequence (PN23) [37]. To describe the performance of the system, the implementation choice for the development of the SCCC decoder will be taken into account. For the other modules, indeed, the parallel GPU implementation has been demonstrated to be enough to optimize the execution of the receiver. The achieved results will be analyzed step-by-step while investigating the added optimization of the code (e.g., Jacobian logarithm approximation, windowing). Regarding the speed performance of the system, it has been compared with the aforementioned SW EGSE [19], but also with two other systems which have been implemented for reference, referred to as *Matlab Script* and *Serial Process*. The Matlab Script is a system implemented on Matlab, modeled on the SW EGSE, but excluding the debug options, to optimize the computation. The serial process was implemented in C++, without exploiting parallel computation. It is the same version of the receiver where each module is executed on a CPU instead of a GPU. As explained before, this implementation has been possible without additional effort, thanks to the use of the Factory Method Design Pattern. Both the Matlab and the C++ solution were executed on the same CPU used for the SDR receiver (i.e., Intel Core i9-12900K).

For the implemented GPU receiver, three different versions have been developed, optimizing the system at each step. In particular, the first system, referred to as *GPU1*, is the first developed system, where the code is optimized only for parallel execution on a GPU. The second system (*GPU2*) has been optimized by an efficient use of the GPU memory and by introducing the approximation of the Jacobian logarithm, while the third system (*GPU3*) has been further optimized by implementing windowing, with a variable number of windows, based on the applied ACM. The three distinct GPU receiver implementations have been compared, demonstrating not only the performance increase over the state-of-the-art receiver, but also the performance improvement across the various implementations. To comprehensively evaluate the speed performance of the system, the NVIDIA CUDA profiling tools have been used [38]. This has allowed for not only the evaluation of the speed of each single developed module, but also for the evaluation of the GPU memory occupancy and read/write operation, to optimize the code.

To evaluate the error correction performance of the system, the BER was computed, applying AWGN impairment to the input and modifying the applied signal-to-noise ratio (Eb_N0) to produce the so-called BER curves. The BER curves show the decrease in the error rate when increasing the Eb_N0, hence the number of symbols with respect to the noise. BER curves have been computed for all the ACM, considering at minimum a number of 1000 PL frames as input, and for all the GPU implementation, to show the implementation loss introduced by optimizing the code. The ideal BER curves are defined in the CCSDS 130.11-G report [36] and are depicted in Figure 6.

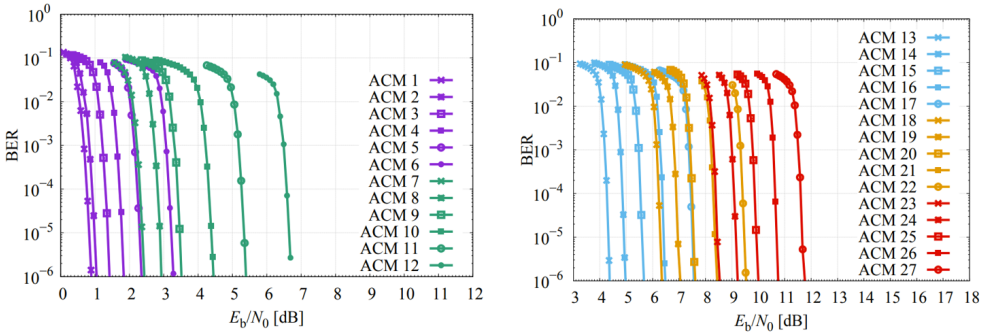

Figure 6. Ideal BER on Linear AWGN Channel for PSK Modulations (**left**) and APSK Modulations (**right**)—as defined in [36].

Results

Table 1 shows the comparison of the measured performance of the SW EGSE, Matlab Script, Serial Process, and GPU implementations for receiving a PL frame, recovering the impairments at front end, and finally producing the output bits related to the first PL frame codeword. The performance was measured on ModCods 1, 15, and 27, which require an increasingly larger number of operations.

Table 1. Execution time comparison to decode 1 Codeword with different ACMs.

Implementation	Time (ms)		
	ACM1	**ACM15**	**ACM27**
SW EGSE	∼3000	∼13,000	∼25,000
Matlab Script	876.15	2182.23	4519.16
Serial Process	31.56	134.32	309.12
GPU1	36.17	143.32	268.55
GPU2	10.17	37.32	68.55
GPU3	1.32	2.88	4.24

ACM 1 is the case that requires the smaller number of operations, proportional to the size of the data. In this case, GPU1 is executed in 36.17 milliseconds for a CW, which represents an improvement of ∼80× with respect to the SW EGSE implementation. ACM 15 and 27 deal with larger sets of data, requiring a larger number of operations. Specifically, for ACM 27, the parallelization introduced in GPU1 produces a performance improvement of ∼90× with respect to the SW EGSE. In general, the GPU1 version is more efficient than both the SW EGSE and the Matlab script, but, for ACM 1 and ACM 15, where smaller sets of data are processed, the serial process on the CPU is faster. This shows how an efficient CPU can actually perform better than a GPU when fewer data are processed. This happens in our case for the ACMs from 1 to 15, hence QPSK, 8PSK and 16APSK. From ACM 16, which is still a 16APSK ModCod, the number of bits per codeword to be processed increases to 25,918. Given the larger datasets, the use of a GPU guarantees an efficiency gain that is then reflected for all the larger ACMs, which require us to process datasets with a number of bits larger than 25,918, up to 43,678 bits for ACM 27. With larger datasets, the GPU SIMD paradigm allows the GPU1 version performance to overcome serial process performance, giving an insight of GPU potential.

The BER curves of the GPU 1 version are shown in Figure 7. We can see that the error decoding performance of this system is similar to the one of the ideal system: the implementation loss introduced in this version is small, with an offset of about 0.2/0.3 dB with respect to the ideal BER curves. This allows us to easily recover the transmitted information bits, also with the possibility of optimizing the code, as has been conducted for the GPU2 and GPU3 versions. The GPU2 and GPU3 naturally add an implementation loss that is reflected on the BER curves computed for these cases.

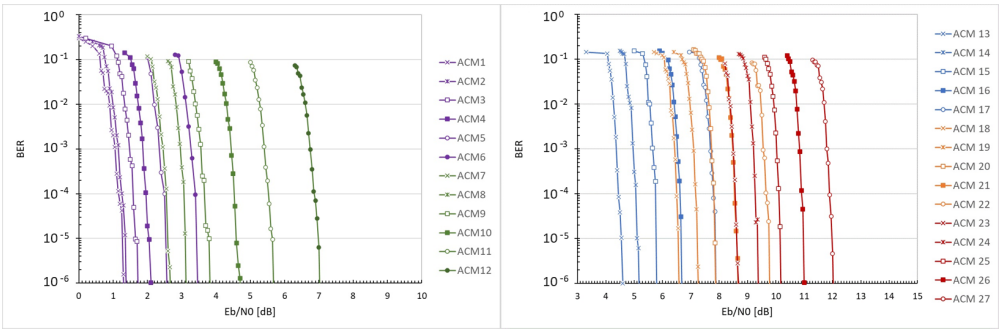

Figure 7. BER curves on linear AWGN channel, drawn for *GPU1* implementation.

Thanks to the optimization mentioned in Section 3.2.2, the GPU2 version, speeding up memory accesses, and implementing a simplified version of the Jacobian logarithm, performs better than the GPU1 version, with a gain of 3× in the case of ACM1, up to ∼4× for the highest ModCod (ACM27). Concerning the SW EGSE, the GPU2 version performance gain is up to ∼360×. Differently from the GPU1, the GPU2 version outperforms the serial process also when processing fewer data, hence for the lower ModCods. In comparison with the GPU1 implementation, the GPU2 system shows an implementation loss increment of about 0.3 dB, as shown in Figure 8. Hence, this optimized version, introducing a 3× performance improvement (in the worst case), suffers an implementation loss of about 0.6 dB. We consider an implementation loss below 1 dB to be acceptable for increasing the performance of the system. So, the loss of the GPU2 version is still considered negligible with respect to the ideal BER values. For example, for ACM 6, the BER drops to 10^{-6} for an $Eb/N0$ value around 3.7 dB, whereas in the ideal curves, the $Eb/N0$ value is about 3.3. In this case, we have an implementation loss of around 0.4 dB. For ACM 22 and ACM 23 (32APSK and 64APSK), instead, the BER value of 10^{-6} is reached with a $Eb/N0$ of around 9 dB; on the other side, the ideal value is around 8.4/8.5 dB, with an implementation loss that increases with respect to lower ACMs.

Finally, the GPU3 version of the code, that is the further-optimized implementation, shows, as expected, the best performances. Concerning the other GPU versions (GPU1 and GPU2), it shows a performance regain of more than 10×, while for ACM 27, it has a gain factor larger than 5500× with respect to the SW EGSE. In comparison with the ideal BER curves, the GPU3 system shows an implementation loss of between 0.7 dB and 0.9 dB, as shown in Figure 9. Overall, we consider the error decoding performance of the system, comprising windowing and simplified max*, to be more than acceptable: the acceptable error recovery performance of a system may vary from case to case and is strongly related to the requirements of the single mission. However, we think that a 1 dB implementation loss can be tolerable for most of the ESA EO missions.

In the end, the proposed solutions have demonstrated to have higher data rate performance than the state-of-the-art SW system, with an implementation loss that is negligible. With respect to the HW solution, the GPU3 version allows us to decode more than 200 CWs per second (in the worst case), while a HW receiver can decode up to 60 k CWs per second. Obviously, the SDR is still much slower than a HW receiver, but, given the flexibility and the low-cost of the system, it can be an acceptable alternative of HW receivers, especially for testing transmitters or for receiving data in science missions. Also, it is important to note that using a higher-performance GPU would provide a proportional performance gain for the implemented receiver, allowing us to speed up the overall execution. Furthermore, given the flexibility of the developed code, and thanks to CUDA framework, the code can be easily re-used on different NVIDIA GPUs or extended to multiple GPUs of the same family, potentially doubling the speed of the system.

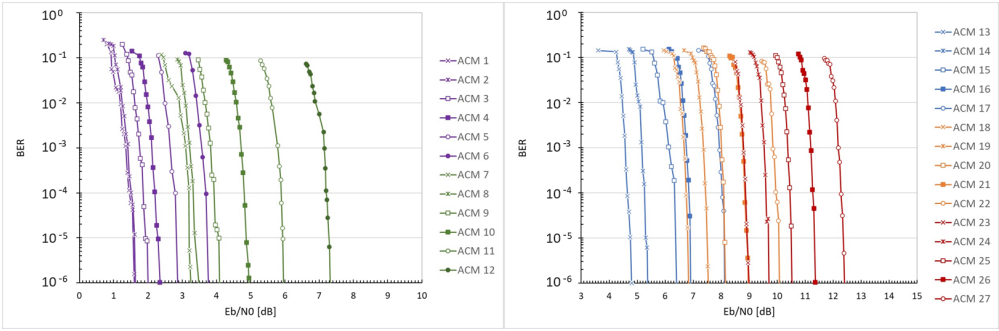

Figure 8. BER curves on linear AWGN channel, drawn for *GPU2* implementation.

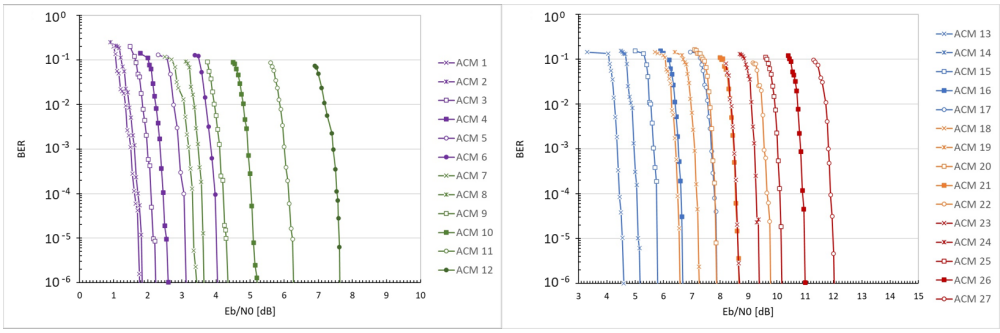

Figure 9. BER curves on linear AWGN channel, drawn for *GPU3* implementation.

5. Conclusions

The recent investments by private companies and start-ups in the space exploration sector have kicked off the so-called New Space Economy. The objective of New Space is to democratize space exploration, making it increasingly accessible to launch products into space. In this scope, the space agencies, and in particular the ESA, are working on the definition of newer communication standards to be adopted in present and future missions, for enhancing the re-usability of deployed systems and consequently providing a positive impact on the economy of space missions. This paper has presented the development of an EGSE for promoting the adoption of the CCSDS 131.2-B standard for telemetry applications in Earth observation missions. This is the standard selected by the ESA for future EO missions. For this reason, it is crucial to dispose of EGSEs compliant with it, for testing and evaluating satellites' transmissions and ground stations' receptions. Also, it is fundamental to provide low-cost systems to boost the use of CCSDS 131.2-B standard, fostering its early adoption in the next-generation missions.

The standard itself is very complex and expensive to implement for both the transmitter and the receiver side, and, at the time of writing, is not used in any mission. To overcome this limitation, the implementation of a low-cost, flexible, SDR receiver based on GP-GPUs has been presented. It aims to provide a cheaper solution with respect to the state-of-the-art receivers, both for on-ground testing of CCSDS 131.2-B compliant transmitters and to be employed in next-generation missions, for receiving data from EO satellites.

The proposed solution has been analyzed, presenting an overview of its structure and the development of the architecture, focusing on the implementation choices made to provide a high-performance solution. Then, the performance of the system has been evaluated, in terms of error recovery and data rate. Overall, the system has been demonstrated to have good performance with respect to other SW-based solutions, setting itself as a valid alternative with respect to the complex and expensive HW system at the state of the art. In fact, it is a flexible system and is easy to deploy and has a much lower cost (a few thousand dollars) with respect to the state-of-the-art HW receivers (hundreds of thousands of dollars).

The presented solution will provide fundamental aid for the adoption of the CCSDS 131.2-B standard for next-generation EO ESA missions. It will promote the standard use by

allowing the easy and low-cost testing of transmitters compliant with it, and by supporting the reception of data for missions where the requirements are relaxed (i.e., science missions). Moreover, the flexibility of the system easily allows for future development, involving the use of more GPUs in parallel, or higher-performance GPUs, increasing the data rate of the system.

Author Contributions: Conceptualization, R.C., G.G. and M.B.; methodology, R.C., G.G. and M.B.; software, R.C.; validation, R.C., G.G. and M.B.; formal analysis, R.C. and G.G.; investigation, R.C. and G.G.; resources, G.G. and M.B.; data curation, R.C., G.G. and M.B.; writing—original draft preparation, R.C.; writing—review and editing, L.F.; visualization, R.C.; supervision, L.F.; project administration, L.F.; funding acquisition, L.F. All authors have read and agreed to the published version of the manuscript.

Funding: This research received no external funding.

Data Availability Statement: Derived data supporting the findings of this study are available from the corresponding author on request.

Acknowledgments: The work presented in this paper has been carried out in collaboration with IngeniArs S.r.l., a SME working in the Aerospace industry.

Conflicts of Interest: Authors Gianluca Giuffrida and Matteo Bertolucci were employed by the company IngeniArs S.r.l. The remaining authors declare that the research was conducted in the absence of any commercial or financial relationship that could be construed as a potential conflict of interest.

Abbreviations

The following abbreviations are used in this manuscript:

ACM	Adaptive Coding and Modulation
AWGN	Additive White Gaussian Noise
BER	Bit Error Rate
CCSDS	Consultative Committee for Space Data Systems
CER	Codeword Error Rate
CPU	Central Processing Unit
CUDA	Compute Unified Device Architecture
CW	CodeWord
ECCS	European Cooperation for Space Standardization
EGSE	Electrical Ground Support Equipment
EO	Earth Observation
ESA	European Space Agency
FD	Frame Descriptor
FER	Frame Error Rate
FM	Frame Marker
GP-GPU	General-Purpose Graphics Processing Unit
HW	Hardware
LEO	Low-Earth Orbit
LLR	Log-Likelihood Ratio
MAP	Maximum A-Posteriori
MHOMS	Modem for High Order Modulation Schemes
ModCod	Modulation and Coding
NewSpace	New Space Economy
PL	Physical Layer
SCCC	Serial Concatenated Convolutional Codes
SDR	Software Defined Radio
SIMD	Single Instruction Multiple Data
SISD	Single Instruction Single Data
SISO	Soft-In Soft-Out
SME	Small Medium-Size Enterprise

SNR	Signal to Noise Ratio
SRRC	Squared-Root Raised Cosine
SW	Software
TF	Transfer Frame
VCM	Variable Coding and Modulation

References

1. Parrella, R.M.; Spirito, G.; Cirina, C.; Falvella, M.C. The New Space Economy and New Business Model. *New Space* **2022**, *10*, 291–297. [CrossRef]
2. Orlova, A.; Nogueira, R.; Chimenti, P. The present and future of the space sector: A business ecosystem approach. *Space Policy* **2020**, *52*, 101374. [CrossRef]
3. SpaceX Website. Available online: https://www.spacex.com (accessed on 25 November 2023).
4. Blue Origin Website. Available online: https://www.blueorigin.com (accessed on 25 November 2023).
5. Virgin Galactic Website. Available online: https://www.virgingalactic.com (accessed on 25 November 2023).
6. Selva, D.; Krejci, D. A survey and assessment of the capabilities of Cubesats for Earth Observation. *Acta Astronaut.* **2012**, *74*, 50–68. [CrossRef]
7. Liddle, J.D.; Holt, A.P.; Jason, S.J.; O'Donnell, K.A.; Stevens, E.J. Space science with CubeSats and nanosatellites. *Nat. Astron.* **2020**, *4*, 1026–1030. [CrossRef]
8. Wekerle, T.; Pessoa, J.B.; Costa, L.E.V.L.; Trabasso, L.G. Status and trends of smallsats and their launch vehicles. An up-to-date review. *J. Aerosp. Technol. Manag.* **2017**, *9*, 269–286. [CrossRef]
9. Fortescue, P.; Swinerd, G.; Stark, J. *Spacecraft Systems Engineering*, 4th ed.; John Wiley & Sons: Hoboken, NJ, USA, 2011.
10. Lange, C.; Grundmann, J.T.; Kretzenbacher, M.; Fischer, P.M. Systematic reuse and platforming: Application examples for enhancing reuse with model-based systems engineering methods in space systems development. *Concurr. Eng.* **2018**, *26*, 77–92. [CrossRef]
11. Baiocco, P. Overview of reusable space systems with a look to technology aspects. *Acta Astronaut.* **2021**, *189*, 10–25. [CrossRef]
12. Consultative Committee for Space Data Systems (CCSDS) Website. Available online: https://public.ccsds.org/default.aspx (accessed on 25 November 2023).
13. European Cooperation for Space Standardization (ECSS) Website. Available online: https://ecss.nl (accessed on 25 November 2023).
14. Cortex High Data Rate (HDR) Receiver for Space Science and Earth Observation. Available online: https://www.safran-group.com/sites/default/files/2021-05/col000016.4.0_cortex_hdr_a4_2.pdf (accessed on 25 November 2023).
15. Ciardi, R.; Giuffrida, G.; Bertolucci, M.; Pagani, E.; Fanucci, L. CCSDS 131.2-B-1 Software Defined Radio receiver featuring GPU accelerators: Up to 1000x with respect to CPU implementation. In Proceedings of the IEEE 9th International Workshop on Tracking, Telemetry and Command Systems for Space Applications (TT&C) ESA-ESTEC, Noordwijk, The Netherlands, 28 November–1 December 2022.
16. Consultative Committee for Space Data Systems (CCSDS). *CCSDS 131.2-B-2 Recommended Standard: Flexible Advanced Coding and Modulation Scheme for High Rate Telemetry Applications*; Consultative Committee for Space Data Systems (CCSDS): Sanford, FL, USA , 2023.
17. Berger, M.; Moreno, J.; Johannessen, J.A.; Levelt, P.F.; Hanssen, R.F. ESA's sentinel missions in support of Earth system science. *Remote. Sens. Environ.* **2012**, *120*, 84–90. [CrossRef]
18. Benedetto, S.; Garello, R.; Montorsi, G.; Berrou, C.; Douillard, C.; Giancristofaro, D.; Ginesi, A.; Giugno, L.; Luise, M. MHOMS: High-speed ACM modem for satellite applications. *IEEE Wirel. Commun.* **2005**, *12*, 66–77. [CrossRef]
19. Diana, L.; Giuffrida, G.; Marini, M.; Cassettari, R.; Davalle, D.; Fanucci, L. SCCC SW EGSE: A software simulator of a satellite downlink communication compliant with the CCSDS 131.2-B-1 standard, with Hardware-In-The-Loop capabilities. In Proceedings of the IEEE 8th International Workshop on Tracking, Telemetry and Command Systems for Space Applications (T&TC), ESA-ESTEC, Noordwijk, The Netherlands, 24–27 September 2019.
20. The MathWorks Inc. MATLAB Website. Available online: https://www.mathworks.com (accessed on 25 November 2023).
21. Ulversoy, T. Software defined radio: Challenges and opportunities. *IEEE Commun. Surv. Tutor.* **2010**, *12*, 531–550. [CrossRef]
22. Akeela, R.; Dezfouli, B. Software-defined Radios: Architecture, state-of-the-art, and challenges. *Comput. Commun.* **2018**, *128*, 106–125. [CrossRef]
23. Kacpura, T.J.; Eddy, W.M.; Smith, C.R.; Liebetreu, J. Software defined radio architecture contributions to next generation space communications. In Proceedings of the 2015 IEEE Aerospace Conference, Big Sky, MT, USA, 7–14 March 2015; pp. 1–12.
24. Pugh, M.; Kuperman, I.; Aguirre, F.; Mojaradi, H.; Spurgers, C.; Kobyashi, M.; Satorius, E.; Jedrey, T. The universal space transponder: A next generation software defined radio. In Proceedings of the 2017 IEEE Aerospace Conference, Big Sky, MT, USA, 4–11 March 2017; pp. 1–14.
25. Arslan, H. *Cognitive Radio, Software Defined Radio, and Adaptive Wireless Systems*; Springer: Dordrecht, The Netherlands, 2007.

26. der Heide, S.V.; Luis, R.S.; Puttnam, B.J.; Rademacher, G.; Koonen, T.; Shinada, S.; Awaji, Y.; Furukawa, H.; Okonkwo, C. 10,000 km Straight-line Transmission using a Real-time Software-defined GPU-Based Receiver. In Proceedings of the 2021 IEEE Optical Fiber Communications Conference and Exhibition (OFC), Washington, DC, USA, 6–11 June 2021; pp. 1–3.
27. NVIDIA QUADRO RTX 4000 Datasheet. Available online: https://www.nvidia.com/content/dam/en-zz/Solutions/design-visualization/quadro-product-literature/quadro-rtx-4000-datasheet-us-nvidia-1060942-r2-web.pdf (accessed on 25 November 2023).
28. Ellis, B.; Stylos, J.; Myers, B. The factory pattern in API design: A usability evaluation. In Proceedings of the 29th International Conference on Software Engineering (ICSE'07), Minneapolis, MN, USA, 20–26 May 2007.
29. Vucetic, B.; Yuan, J. *Turbo Codes: Principles and Applications*; Springer Science & Business Media: New York, NY, USA, 2012; Volume 559.
30. Huber, K. Some comments on Zech's logarithms. *IEEE Trans. Inf. Theory* **1990**, *36*, 946–950. [CrossRef]
31. Bahl, L.; Cocke, J.; Jelinek, F.; Raviv, J. Optimal decoding of linear codes for minimizing symbol error rate (corresp.). *IEEE Trans. Inf. Theory* **1975**, *20*, 284–287. [CrossRef]
32. Boutillon, E.; Douillard, C.; Montorsi, G. Iterative decoding of concatenated convolutional codes: Implementation issues. *Proc. IEEE* **2007**, *95*, 1201–1227. [CrossRef]
33. Abrantes, A.S. *From BCJR to Turbo Decoding: MAP Algorithms Made Easier*; Faculdade de Engenharia da Universidade do Porto (FEUP): Porto, Portugal, 2004.
34. JGross, W.; Gulak, P.G. Simplified MAP algorithm suitable for implementation of turbo decoders. *Electron. Lett.* **1998**, *34*, 1577–1578. [CrossRef]
35. Cheng, J.; Ottosson, T. Linearly approximated log-MAP algorithms for turbo decoding. In Proceedings of the IEEE 51st Vehicular Technology Conference, Tokyo, Japan, 15–18 May 2000; Volume 3, pp. 2252–2256.
36. Consultative Committee for Space Data Systems (CCSDS). *CCSDS 131.11-G-2 Informational Report: SCCC—Summary of Definition and Performance*; Consultative Committee for Space Data Systems (CCSDS): Sanford, FL, USA, 2023.
37. CCITT. *Specification on Measuring Equipment—Digital Test Patterns for Performance Measurements on Digital Transmission Equipment*; International Telecommunication Union: Geneva, Switzerland, 1992.
38. Bridges, R.A.; Imam, N.; Mintz, T.M. Understanding GPU power: A survey of profiling, modeling, and simulation methods. *ACM Comput. Surv.* **2016**, *49*, 1–27. [CrossRef]