

Article

Leveraging Seed Generation for Efficient Hardware Acceleration of Lossless Compression of Remotely Sensed Hyperspectral Images

Amal Altamimi ^{1,2} and Belgacem Ben Youssef ^{1,*} 

¹ Department of Computer Engineering, College of Computer and Information Sciences, King Saud University, P.O. Box 51178, Riyadh 11543, Saudi Arabia

² Space Technologies Institute, King Abdulaziz City for Science and Technology, P.O. Box 8612, Riyadh 12354, Saudi Arabia; aaltamimi@kacst.gov.sa

* Correspondence: bbenyoussef@ksu.edu.sa

Abstract: In the field of satellite imaging, effectively managing the enormous volumes of data from remotely sensed hyperspectral images presents significant challenges due to the limited bandwidth and power available in spaceborne systems. In this paper, we describe the hardware acceleration of a highly efficient lossless compression algorithm, specifically designed for real-time hyperspectral image processing on FPGA platforms. The algorithm utilizes an innovative seed generation method for square root calculations to significantly boost data throughput and reduce energy consumption, both of which represent key factors in satellite operations. When implemented on the Cyclone V FPGA, our method achieves a notable operational throughput of 1598.67 Mega Samples per second (MSps) and maintains a power requirement of under 1 Watt, leading to an efficiency rate of 1829.1 MSps/Watt. A comparative analysis with existing and related state-of-the-art implementations confirms that our system surpasses conventional performance standards, thus facilitating the efficient processing of large-scale hyperspectral datasets, especially in environments where throughput and low energy consumption are prioritized.



Citation: Altamimi, A.; Ben Youssef, B. Leveraging Seed Generation for Efficient Hardware Acceleration of Lossless Compression of Remotely Sensed Hyperspectral Images. *Electronics* **2024**, *13*, 2164. <https://doi.org/10.3390/electronics13112164>

Academic Editors: Dimitrios Dechouniotis, Marios Avgeris, Konstantinos Tsitsekis and Vitoropoulou Margarita

Received: 1 May 2024

Revised: 21 May 2024

Accepted: 29 May 2024

Published: 1 June 2024



Copyright: © 2024 by the authors. Licensee MDPI, Basel, Switzerland. This article is an open access article distributed under the terms and conditions of the Creative Commons Attribution (CC BY) license (<https://creativecommons.org/licenses/by/4.0/>).

Keywords: hardware acceleration; lossless compression; FPGA; hyperspectral image; seed generation; efficiency; throughput; power

1. Introduction

Hyperspectral imaging, capturing data across hundreds of spectral wavelengths, has significantly expanded the capabilities of remote sensing, influencing fields such as environmental science [1,2], agriculture [3–5], and defense [6]. The richness of hyperspectral data allows for unprecedented levels of detail in observing the Earth's surface. However, it also results in enormous data volumes that challenge existing processing, storage, and transmission capacities.

To address these challenges, compression is essential to reduce the data size, facilitating faster and feasible data transmission within the constraints of available bandwidth. It also allows for more efficient use of storage, reducing both physical space requirements and associated costs [7]. In particular, lossless compression ensures data integrity and analytical accuracy. This is vital for applications requiring precise data analysis, such as environmental monitoring and scientific research. Additionally, lossless compression provides flexibility, ensuring data can be reanalyzed as new techniques are developed.

Moreover, hardware acceleration plays a pivotal role in optimizing the process of compression for hyperspectral images. Utilizing specialized hardware components such as field-programmable gate arrays (FPGAs), hardware acceleration significantly speeds up data processing tasks. This acceleration is critical for real-time applications where rapid data processing is essential. Moreover, the reduced power consumption achieved through

hardware acceleration is particularly advantageous for platforms with limited energy resources [8]. FPGAs stand out due to their reconfigurable nature and ability to handle large data volumes promptly through parallel processing. Their low power consumption also fits the stringent energy requirements of spaceborne systems. As such, FPGAs provide adaptable, robust solutions for diverse imaging system requirements, further optimizing onboard resource utilization and operational efficiency [9].

Given these advancements, maintaining a focus on reducing power requirements is essential, especially for spaceborne systems and unmanned aerial vehicles [10,11]. This involves refining hardware components in addition to exploring energy-efficient algorithms and architectures that reduce the overall power footprint, thus extending the operational lifespan of imaging systems. Looking ahead, the field of hyperspectral imaging faces several challenges and opportunities that will likely influence its development. One major challenge is the sheer volume of data generated by modern hyperspectral sensors, which can rapidly overwhelm existing storage and transmission infrastructure. Addressing this will require the development of innovative solutions for both compression algorithms and hardware acceleration techniques [12].

In response to these challenges, this study harnesses FPGA technology to significantly enhance existing data compression capabilities. Below, we detail the key contributions of our research:

- We describe the hardware acceleration of a lossless compression algorithm designed specifically for hyperspectral images. Optimized for FPGA platforms and enhanced by leveraging seed generation techniques, this adaptation demonstrates practical applicability and effectiveness in real-time processing.
- We present the implementation of the lossless algorithm targeting an FPGA with modest capabilities, such as Cyclone V. This implementation demonstrates substantial performance improvements, achieving a throughput of 1598.67 Mega Samples per second (MSps), while maintaining a power requirement below 1 Watt. This highlights the durability of the designed algorithm and the optimized hardware acceleration in achieving both computational speed and efficiency.

The rest of the paper is structured as follows: Section 2 gives a short review of some recent works related to the hardware acceleration of lossless compression of hyperspectral images (HSIs). Then, Section 3 describes the optimized hardware implementation of lossless compression of remotely sensed hyperspectral images utilizing a seed generation approach. This is followed by the experimental and performance results of the said hardware acceleration targeting a Cyclone V FPGA board and its comparison to other state-of-the-art implementations in Section 4. Finally, our concluding remarks and future work are provided in Sections 5 and 6, respectively.

2. Related Work

Recent advancements in hardware-accelerated hyperspectral image compression have significantly enhanced the capabilities of spaceborne imaging systems by facilitating the rapid and efficient processing and transmission of complex data. Building upon our systematic review in this area [8], which covered works up to mid-May of 2021, we explore recent research contributions that complement our previous review. The selection criteria for these studies remain consistent with those applied in our earlier work, with the inclusion criterion specifically focused on research published after this date. The studies selected for this review are specifically aimed at optimizing both performance and resource efficiency in lossless compression systems.

The work authored by Lili Zhang et al. examines the development of a hyperspectral image compression system tailored for satellite data transmission [13]. It highlights the utilization of the xc7k325tffg900 FPGA chip from Xilinx (San Jose, CA, USA), aiming to optimize the system for improved calculation speed, compression efficiency, and error resilience. The study details the implementation of the CCSDS 123.0-B-1 lossless compression algorithm [14], featuring a 3D space adaptive linear prediction and adaptive Rice

encoding. Although specific performance metrics are not reported, such as compression ratio (CR) and power consumption, the study emphasizes substantial enhancements in error handling and system resource utilization, demonstrating a robust design tailored to meet the stringent demands of spaceborne image data compression.

In a similar context, another study explores the development and optimization of a low-cost hardware accelerator that complies with the CCSDS 123.0-B-2 standard for lossless hyperspectral image compression [15], aiming to enhance spaceborne image processing [16]. It details the implementation using both high-level synthesis (HLS) and hardware description language (HDL), comparing their performance, throughput, and power consumption. The HLS implementation demonstrated a throughput of up to 9.38 Mega Samples per second (MSps) with power reductions achieved through optimization techniques, while the HDL version achieved higher throughput at 21.47 MSps, emphasizing its efficiency for real-time space applications. The paper also highlights design choices that minimize resource utilization and improve reliability against space-related effects on circuits. As part of future work, it suggests the integration of these accelerators into multi-core satellite systems for enhanced performance.

Continuing with other FPGA implementations, a recent comprehensive study explores a real-time FPGA implementation of the CCSDS 123.0-B-2 standard for compressing hyperspectral images [15,17]. Employing a deeply pipelined FPGA architecture and a novel sample ordering method called frame interleaved by diagonal (FID), the implementation achieves significant enhancements in processing speed, achieving a throughput of 249.6 million samples per second at a power consumption of only 1.21 watts. This design, developed in VHDL and tested on a Virtex-7 VC709 FPGA board, focuses on maintaining real-time processing capabilities while ensuring efficient use of hardware resources, occupying between 14% and 50% depending on image size. The hybrid coder component allows for both lossless and lossy compression, adapting dynamically to the compression needs while optimizing the trade-offs between compression ratio and image integrity.

Moreover, Chatziantoniou et al. discuss the development and implementation of a high-throughput hybrid entropy coder based on the CCSDS 123.0-B-2 standard [15], specifically designed for space-grade SRAM FPGA technology [18]. The architecture utilizes a systolic design pattern to ensure modularity and latency insensitivity, achieving a consistent throughput of 1 sample/cycle with minimal FPGA resource usage. Implemented and tested on a Xilinx KCU105 development board with a Xilinx Kintex Ultrascale XCKU040 SRAM FPGA, the system is also compatible with the Xilinx Radiation Tolerant Kintex UltraScale XQRKU060 devices. Some of the key aspects of the implementation include its integration with the state-of-the-art SpaceFibre serial link interface for emulation of on-board deployment, achieving a throughput of 305 MSps and a power consumption of 1.525 Watts.

Lastly, a study investigates the parallelization of the CCSDS multispectral and hyperspectral image compression (CCSDS-MHC) algorithm described in CCSDS 123.0-B-1 using OpenMP to enhance execution times in the processing of hyperspectral images [14,19]. Initially, the algorithm is adapted into a C/C++ program to include both compression and decompression functionalities. Through identifying parallelizable sections and applying OpenMP directives, the study successfully demonstrates significant improvements in execution speed across different multicore systems by processing image bands concurrently. The effectiveness of this approach is validated on various hardware setups, showing considerable speedups and consistent compression ratios, proving the potential of parallel processing in real-world satellite imaging applications where fast data handling is crucial. Table 1 summarizes the main findings from each of the previous studies, including important performance metrics and implementation strategies.

Table 1. Overview of recent studies on hardware acceleration of lossless compression of remotely-sensed hyperspectral images.

Compression Method	CR	Throughput (MSps)	Power (Watts)	Efficiency (MSps/W)	Reference, Year
CCSDS 123.0-B-1	-	-	-	-	[13], 2021
CCSDS 123.0-B-2	-	21.47	-	-	[16], 2023
CCSDS 123.0-B-3	-	9.38	0.149	63	[16], 2023
CCSDS 123.0-B-2	-	249.6	1.21	206.3	[17], 2022
CCSDS-123.0-B-2	-	305	1.525	200	[18], 2022
CCSDS-MHC	3.28	-	-	-	[19], 2022

Collectively, the frequent adoption of FPGA technology in the studies reviewed, with the exception of one, emphasize its significant advantages in terms of flexibility, reconfigurability, and efficient handling of parallel computations, which are critical for the stringent demands of remote sensing applications. These studies mainly focus on optimizing power consumption and throughput rather than compression ratio, a decision likely influenced by the operational constraints and the critical need for efficiency in remote sensing environments where power is limited, and data must be processed rapidly and reliably. The use of the CCSDS standard across these studies further aligns with this focus, as it provides well-established guidelines that ensure reliability and efficiency in data compression and transmission for remote sensing systems.

Pipelining within FPGA architectures plays a crucial role here, enhancing throughput significantly by allowing multiple data processing stages to operate simultaneously, which optimizes the flow of data through the compressor and effectively multiplies the processing capacity without a corresponding increase in power usage. On the other hand, the use of high-level synthesis (HLS) tools, despite their potential for simplifying the design process, can sometimes be detrimental to performance. Despite advances in AI and machine learning that promise more adaptive and powerful compression techniques, HLS tools have not yet fully bridged the gap in efficiently translating these complex algorithms into highly optimized hardware implementations. This highlights a critical area of ongoing research and development in the field of FPGA-based system design.

In conclusion, these reviewed studies highlight many significant advancements in FPGA implementations and parallel processing for hyperspectral image compression, particularly in remote sensing applications. Despite these developments, gaps in performance metrics persist, pointing to the importance of continued progress in this area and to potentially valuable directions for future research.

3. Materials and Methods

This section reviews our previously proposed method for lossless compression [20], primarily designed for hyperspectral data by leveraging a novel seed generation technique for efficient square root calculation. Moving forward, we then shift our focus to the hardware implementation of this compression system, employing an FPGA platform optimized for power and real-time processing. Detailed design considerations and performance metrics will be discussed to highlight the practical application of this system in pertinent operational environments.

3.1. Lossless Compression

Compression is achieved by exploiting the mathematical properties of the square root, combined with the capabilities of entropy encoding techniques. The reduction in data

size is facilitated by recognizing that the integer part of the square root of a number x requires approximately half the bits needed for x , specifically $\lceil n/2 \rceil$ bits, where n is the number of bits in the binary representation of x . This reduced bit requirement contributes to lower entropy, as it translates to less randomness and higher predictability in the dataset. Consequently, by employing a straightforward entropy encoder, the compression process is significantly optimized.

Although the square root operation aids in compression, it remains one of the most computationally intensive operations due to the complexity of its algorithms [21]. Generally, square rooting methods can be classified into subtractive, multiplicative (also known as iterative), and approximation methods. A selection of square rooting algorithms is examined in [22], with the least complex method requiring 34 clock cycles. While this is relatively low cost for a square root operation, it is considered high when compared to simpler arithmetic operations, such as addition or comparison, which require only one clock cycle for most architectures [23]. On the other hand, bit manipulation techniques provide a rough approximation of the square root value with significantly fewer clock cycles. These techniques exploit properties of the binary representation to perform tasks such as counting the leading or trailing zeros, extracting contiguous bits, and locating the first or last set bit, among others. Bit manipulation techniques are primarily used to generate a rough estimate of the square root as a seed for iterative square rooting methods [24]. The accuracy of these initial seeds is crucial, as a more precise seed reduces the number of iterations needed to calculate the square root accurately. The effectiveness of bit manipulation techniques in generating accurate seeds is also investigated in [22], with our proposed technique for seed generation demonstrating the highest accuracy. The main advantage of this approach lies in its low complexity employing simple arithmetic operations, making it suitable for real-time compression. Building on this foundation, we employ the seed generation technique for data reduction, as it offers both low complexity and the accuracy required for this specific application.

3.1.1. Preprocessing

The preprocessing stage involves decorrelating hyperspectral data to allow for more streamlined compression. This is achieved by employing a bitwise exclusive or (XOR) operation [25]. Typically, correlated data, such as hyperspectral images, exhibit similar values in their most significant bits. Therefore, the use of the XOR operation sets the most significant bits into zeros, resulting in a lower data entropy while maintaining unsigned integers. This is achieved by XORing the adjacent bands B_i of each line of the acquired scene, except for the first band B_0 , as shown in Equation (1) next. Subsequently, the original data can be reconstructed at the decoder by repeating the XOR operation starting from the first band. Hence, we have the following formulae:

$$B_i = B_i \oplus B_{i-1}, \text{ for } i > 0 \quad (1)$$

where B_i denotes the decorrelated band and $\oplus$ indicates the XOR operation. This process mirrors the technique used in cryptography, where encrypting a message involves XORing it with a key, and decrypting it involves XORing the encrypted message with the same key, showcasing the role of the XOR logic in both data security and recovery.

3.1.2. Computation of the Integral Part

The primary functionality of the compression system relies on the computation of the integer square root. The initial estimate of the square root of a value x is derived from the seed s_0 which is computed using bit manipulation techniques. Specifically, s_0 is determined by averaging the most significant half (MSH) of the binary representation of x and the term $2^{\lceil n/2 \rceil}$, where n is the number of bits in x , which together estimate the integer square root value:

$$s_0 = 0.5 \times (\text{MSH} + 2^{\lceil n/2 \rceil}). \quad (2)$$

Processing decorrelated data in byte chunks improves compression ratios by frequently resulting in entirely zero bytes, which are then shortened using a run-length encoder. Additionally, processing byte-sized segments helps avoid estimation errors in square root calculations associated with larger chunks. Error analysis of square root estimation via seed generation, detailed in [22], shows that deviations from the correct integer square root start at 9-bit numbers. Thus, processing data in byte chunks ensures accurate calculation of the integer square root.

3.1.3. Calculation of the Binary Logarithm

For the computation of the seed value, it is essential to identify n to accurately determine the shift amount needed, as depicted in Equation (2). One approach to compute n is by adding one to the floor value of the calculated $\log_2 x$. Mansour et al. provide insight into the most commonly used algorithms for computing the binary logarithm [26]. One approach involves precomputing and storing logarithmic values in a table for quick look up during calculations. Another approach is the iterative method that refines the estimate using Equation (3) by breaking down the input x into a mantissa m and an exponent e :

$$\log_2(x) = e + \log_2(m). \quad (3)$$

In addition, the coordinate rotation digital computer (CORDIC) algorithm that uses simple shift and add operations to efficiently compute logarithms. It calculates $\ln(m)$ using the CORDIC process, given by Equation (4), and then converts it to a base 2 logarithm by employing Equation (5).

$$\ln(m) = 2 \cdot \tanh^{-1} \left(\frac{m-1}{m+1} \right), \quad (4)$$

$$\log_2(m) = \frac{\ln(m)}{\ln(2)}. \quad (5)$$

Lastly, a method utilizes Taylor series expansion to approximate logarithmic values, with accuracy improving as more k -terms are added to the series. By expanding $\ln(x)$ using the series given by Equation (6), the binary logarithm is then obtained as $\log_2(x) = \ln(x)$:

$$\ln(x) = \sum_{k=1}^{\infty} \frac{(-1)^{k-1}}{k} (x-1)^k. \quad (6)$$

The integer binary logarithm can also be obtained utilizing bit manipulation techniques such as counting the number of leading zeros of the unsigned binary representation of x , i.e., by locating the first set bit. Many hardware platforms offer support for equivalent operations, which expedite the process of finding the binary logarithm [27]. These techniques typically have a complexity of $O(n)$, where n represents the number of bits required to represent the value of x . Alternatively, utilizing binary search to determine the $\log_2 x$ value incurs a complexity of $O(\log_2 x)$ in the worst-case scenario [28].

3.1.4. Computing the Fractional Part

To losslessly achieve this reduction, it is essential to preserve the fractional part of the square root for accurate retrieval of the original value of x . To achieve this, we utilize the fact that there are $2s_i$ integers between any two consecutive square integers s_i^2 and s_{i+1}^2 . Therefore, the fractional part can be encoded as the distance of x from the nearest square number s_i^2 , where $s_i^2 < x$. To uniquely encode the fractions that correspond to each integer square root s_i , we need $\lceil \log_2 2s_i \rceil$ bits, based on the aforementioned observation. Consequently, as the value of the square root s_i decreases, as result of the decorrelation step, the number of bits in the fractional part also decreases, which yields more reduction.

To ensure accurate data retrieval, special considerations are taken when encoding the fractional part, particularly when the integer square roots are powers of two, such as 1, 2, 4, and 8. In these cases, an additional bit is required to the standard calculation of $\lceil \log_2 2s_i \rceil$ in order to accommodate all fractions. To ensure that the bit lengths for the fractional parts remain consistent, these power-of-two squares are represented by a sequence of four zeros in their integral part. This pattern signals the decoder to interpret subsequent bits using unary coding within the predefined bit length. For example, the fractional parts of 2, 4, and 8 are 10, 110, and 1110, respectively, while the integral part for all these cases is represented as 0000. This means that for a seed in the form of 2^m , m represents the number of ones in the unary code of the fractional part. For instance, for $s_i = 4 = 2^2$, $m = 2$ and the corresponding unary code of the fractional part is 110. If the fractional value is not zero, the block adjusts the fractional output by subtracting one, thereby accommodating all possible fractions. Table 2 lists the codewords, ensuring uniform encoding scheme.

Table 2. Codewords generated by the uniform encoding block employed to ensure a consistent encoding scheme.

x	s_0	Integral Part (Seed)	Fractional Part
1	1	0000	0 (unary)
2	1	0001	0
3	1	0001	1
4	2	0000	10 (unary)
5	2	0010	00
6	2	0010	01
7	2	0010	10
8	2	0010	11
9	3	0011	00
...	...	...	...
16	4	0000	110 (unary)
17	4	0100	000
18	4	0100	001
19	4	0100	010
20	4	0100	011
21	4	0100	100
22	4	0100	101
23	4	0100	110
24	4	0100	111
25	5	0101	0000
...	...	...	...

3.1.5. Postprocessing

The final step aims to map the most frequently occurring seed values to Rice codes that use fewer bits, thereby reducing the overall data volume. This mapping strategy, developed based on prior observations that seed values maintain a consistent distribution for the same imager type, and therefore, is performed offline. To support rapid and efficient compression, a compact lookup table is employed. This table is intentionally restricted to just 18 bytes, which is sufficient to cover all necessary variations of the 8-bit seed values with the maximum Rice code of nine bits. This efficient setup guarantees quick data retrieval in constant time $O(1)$, effectively optimizing compression performance. The flowchart of the HSI lossless compressor summarizing the aforementioned steps is given below in Figure 1.

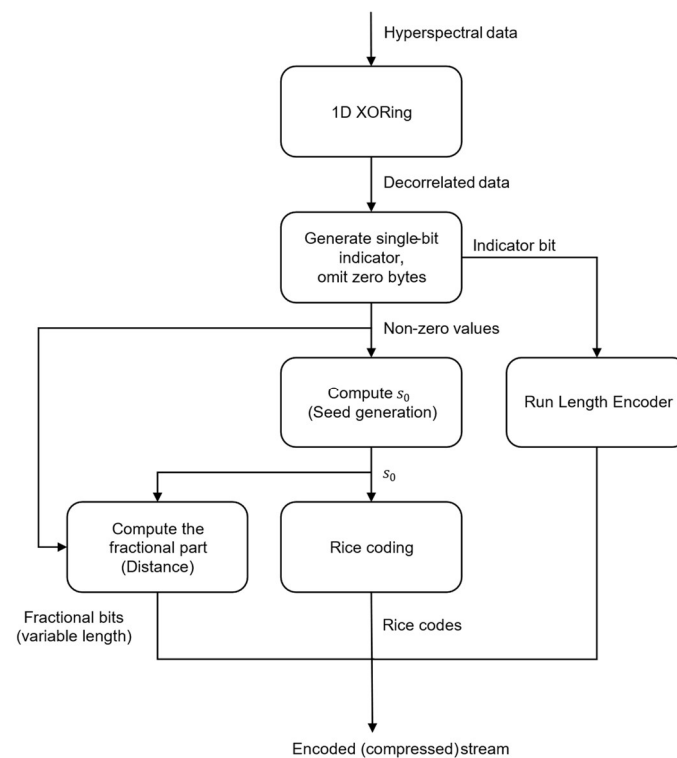


Figure 1. The HSI lossless compressor contains three main stages: (1) preprocessing using one-dimensional XORing; (2) computing s_0 based on the seed generation method; and (3) postprocessing of the fixed length integral part using Rice codes.

3.2. Hardware Implementation

In this section, we provide a thorough examination of the hardware structures and logical constructs of our compression engine, as depicted previously in Figure 1. Illustrated with schematic diagrams and supported by detailed algorithmic descriptions, this section provides the requisite depictions of the internal logic of the implemented hardware to convey a deeper understanding of its working details.

The selected logic is intended to minimize complexity and the number of operations, using a well-pipelined design to enhance processing efficiency and throughput. Notably, this design avoids using memory blocks and heavily relies on I/O operations, allowing for streamlined data handling and reduced latency. In addition, for certain blocks in our synchronous design, we choose to mirror the input to the output for several reasons. First, it maintains timing alignment, ensuring that data arrives at downstream components precisely when needed. It also simplifies debugging by allowing verification of data at various stages and optimizing resources by avoiding unnecessary storage or buffering.

3.2.1. Bitwise XOR Logic

The initial component of our system involves a bitwise XOR operation applied to 8-bit inputs, which is essential for decorrelating the data in preparation subsequent processing stages. This XOR logic is executed by comparing the current input with a delayed version of itself, achieved through a one-clock-cycle delay managed by a register initially set to zero. This setup ensures that the first input remains unchanged, preserving the integrity of the data at the beginning of processing. Figure 2 below displays the register transfer level (RTL) representation of the XOR block. On each rising edge of the clock, the input data is captured into the register, and an XOR operation is performed between this data and its delayed counterpart. The output from this operation, termed XORed, feeds into the next block to determine the necessary shift amount for the seed generation.

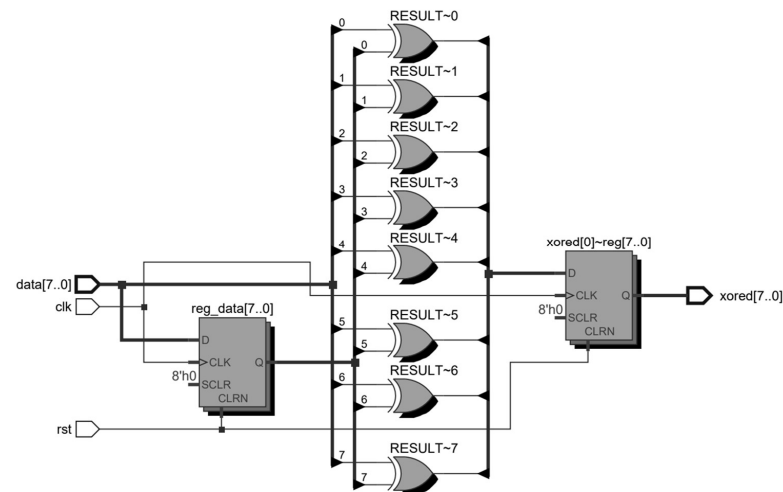


Figure 2. RTL graphical representation of the bitwise XOR logic.

3.2.2. Shift Amount Calculator

The subsequent block calculates the shift amount necessary for seed computation, represented as $\lfloor n/2 \rfloor$. This can be achieved by computing the minimum number of bits, n , required to represent the XORed value, followed by a single right shift operation to derive the shift amount. This can be realized by one of two approaches: One involves finding n by utilizing a binary search followed by a single right shift operation, as outlined in Algorithm 1. The other directly computes the shift amount through a modified binary search process, as detailed in Algorithm 2.

Algorithm 1: Binary search to determine shift_amount based on the value of XORed

Input: XORed

Output: shift_amount

1. Determine n based on the value of XORed using nested comparisons:

```

if XORed > 15
    if XORed > 63
        if XORed > 127
             $n \leftarrow 8$  //XORed is greater than 127
        else
             $n \leftarrow 7$  //XORed is between 64 and 127
    else
        if XORed > 31
             $n \leftarrow 6$  //XORed is between 32 and 63
        else
             $n \leftarrow 5$  //XORed is between 16 and 31
else
    if XORed > 3
        if XORed > 7
             $n \leftarrow 4$  //XORed is between 8 and 15
        else
             $n \leftarrow 3$  //XORed is between 4 and 7
    else
        if XORed > 1
             $n \leftarrow 2$  //XORed is between 2 and 3
        else
             $n \leftarrow 1$  //XORed is 1 or less

```

2. Compute shift_amount as half of n by right-shifting n by 1:

```

shift_amount  $\leftarrow n \gg 1$ 

```

3. Return shift_amount

Algorithm 2: Modified binary search to determine shift_amount based on the value of XORed**Input:** XORed**Output:** shift_amount**1. Determine shift_amount based on the value of XORed using nested comparisons:**

```

if XORed > 31
    if XORed > 127
        shift_amount ← 4          //XORed is greater than 127
    else
        shift_amount ← 3          //XORed is between 32 and 127
else
    if XORed > 7
        shift_amount ← 2          //XORed is between 8 and 31
    else
        if XORed > 1
            shift_amount ← 1      //XORed is between 2 and 7
        else
            shift_amount ← 0      //XORed is 1 or less

```

2. Return shift_amount

Algorithm 2 shortens the number of nested if statements of the binary search leading to a reduction of one operation. Nonetheless, the complexity of the binary search for both algorithms remains the same as $O(\log_2 n)$. The RTL graphical representation is provided in Figure 3, depicting the data flow and logic gates involved. Table 3 below illustrates the rationale behind the choice of pivots in the binary search. These pivots serve as the point of comparison to divide the search into smaller segments.

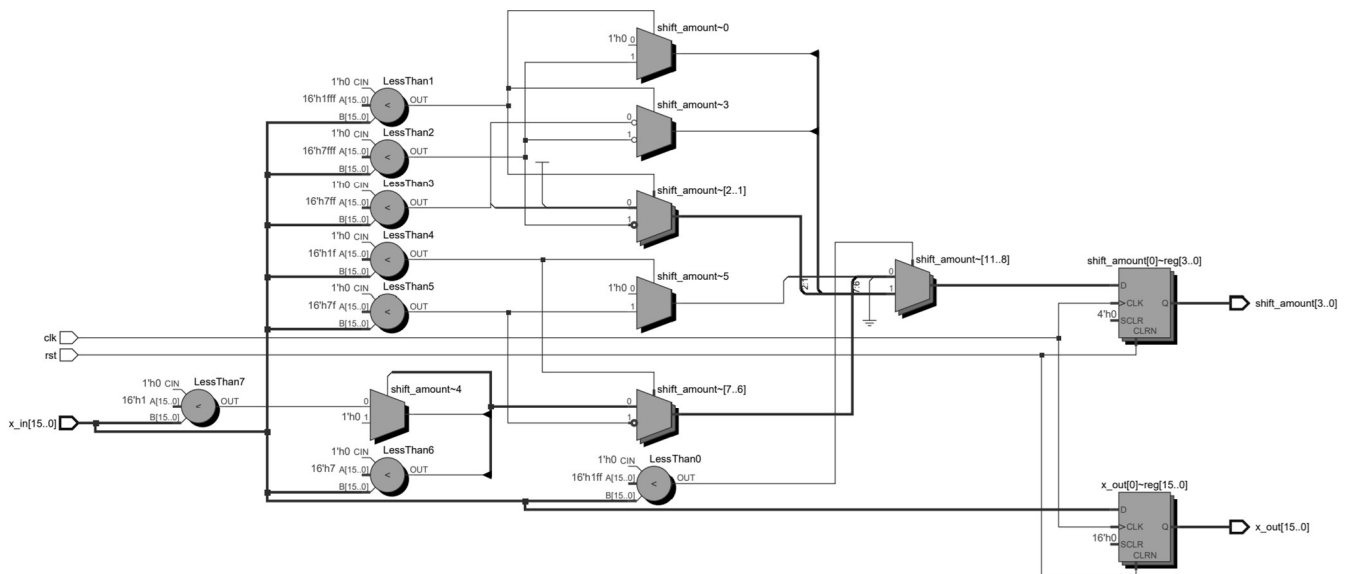


Figure 3. RTL graphical representation of the shift amount calculator.

The resulting shift amount from this block is essential for the seed generation described next.

Table 3. The maximum value of x for each length n of the binary representation and the corresponding shift amounts $\lfloor n/2 \rfloor$ used to guide the binary search algorithm.

Number of bits (n)	Pivot ¹	$\lfloor n/2 \rfloor$
1	1	0
2	3	1
3	7	1
4	15	2
5	31	2
6	63	3
7	127	3
8	255	4

¹ The maximum possible value using n bits.

3.2.3. Seed Generation Logic

The seed generation block operates by processing an 8-bit XORed value as input to generate its corresponding seed value, based on the specified shift amount. Upon each rising edge of the clock, the block first right shifts the XORed value by the shift amount to extract the most significant half, MSH. Concurrently, it calculates the scaled base $2^{\lfloor n/2 \rfloor}$ by left-shifting the constant (1) by the same amount. The latter operation strictly adheres to Equation (2). This is because raising the constant 2 to a given power, say k , is mathematically equivalent to multiplying 1 by 2, k times. In binary representation, this is effectively the same as left-shifting the binary number 1 by k positions. Next, the two terms are then added, and their sum is right-shifted and output as the seed. Since the XORed value has a maximum of 8 bits, the seed value is ensured not to exceed 4 bits. As depicted in Figure 4, the seed generation logic also produces a mirrored output of the XORed value to be retained for further processing stages.

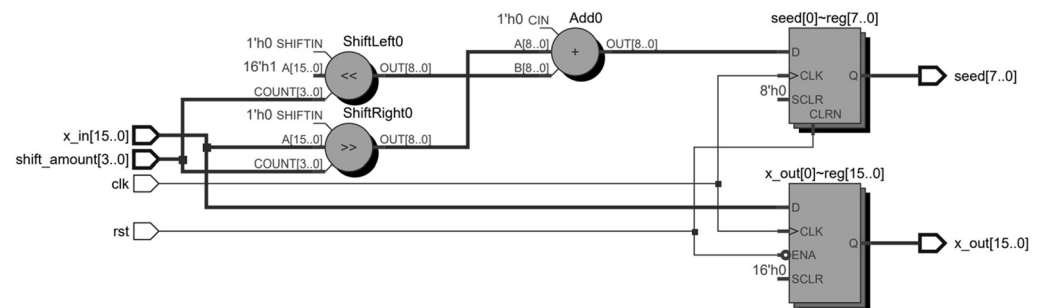


Figure 4. RTL graphical representation of the seed generation logic.

3.2.4. Fraction Calculation Logic

The logic block processes an 8-bit XORed input, which results from a prior XOR operation, along with a 4-bit seed input, to estimate the fractional part of a square root. Operating on a clock signal, the logic first squares the seed to compute the expected square value, SQ. It then evaluates whether the XORed value is less than SQ. If so, the block adjusts the seed by decrementing it, recalculates the square, and computes the difference from the XORed value. If not, it directly calculates the difference, maintaining the relationship that “the fractional part can be encoded as the distance of x from the nearest square number s_i^2 , where $s_i^2 < x$ ” [24]. This difference is then used to determine the fractional output by taking the least significant five bits. The adjusted, or unaltered, seed value is output, while XORed value is mirrored as an output for further processing. Figure 5 presents an illustration of the RTL configuration for the fraction calculation logic.

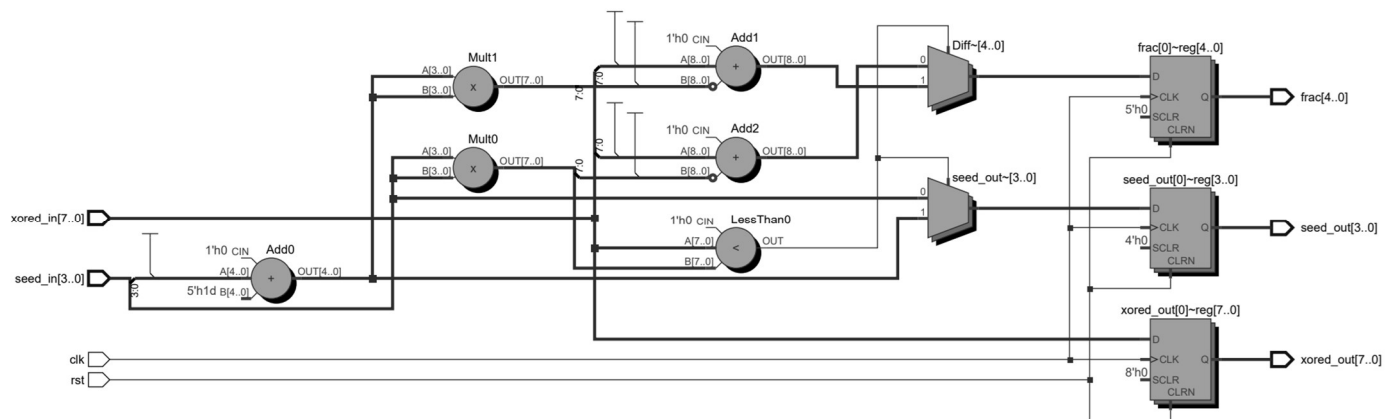


Figure 5. RTL graphical representation of the fraction calculation logic.

3.2.5. Uniform Encoding Block

This block prevents the addition of an extra bit to certain fractional parts to ensure consistent encoding. During each clock cycle, if the fractional part is zero, the code checks if the seed is a power of two (1, 2, or 4) and sets the output fractional part to a predefined value while zeroing the seed output. For non-zero fractional parts, it adjusts the fractional output by subtracting one, ensuring accurate representation and consistency in the encoding scheme. This logic allows the decoder to interpret subsequent bits using unary coding, maintaining uniform bit lengths and facilitating accurate data retrieval. Figure 6 depicts the adjustment undertaken to ensure that both outputs reflect the necessary modifications, thus maintaining the integrity of the encoding process.

3.2.6. Direct Rice Encoder

This block follows the uniform encoding block in the pipeline to capture the updated value of the seed that is of the integral part. Since there are only 16 possible values for the 4-bit input seed, direct mapping of Rice codes is used. A case statement maps each seed value to its respective Rice code. The concise structure of the Rice encoder is depicted in Figure 7.

The output of this encoder corresponds to the second structure of the compressed stream. We note here that the fractional parts require no further processing and comprise the third structure of the compressed stream.

3.2.7. Zero Detection Logic

The zero-detection logic is a straightforward component designed to keep track of the omitted zero bytes to reduce the overall volume while maintaining data integrity. It operates on each rising edge of the clock signal, evaluating the XORed value. If it is entirely set to zeros, the output signal remains low; conversely, if any non-zero values are detected, the output is set to high. The output of this block feeds into a run-length encoder, which compresses runs of similar values, further enhancing the compression performance. This functionality is visually represented through an RTL graphical representation in Figure 8 below.

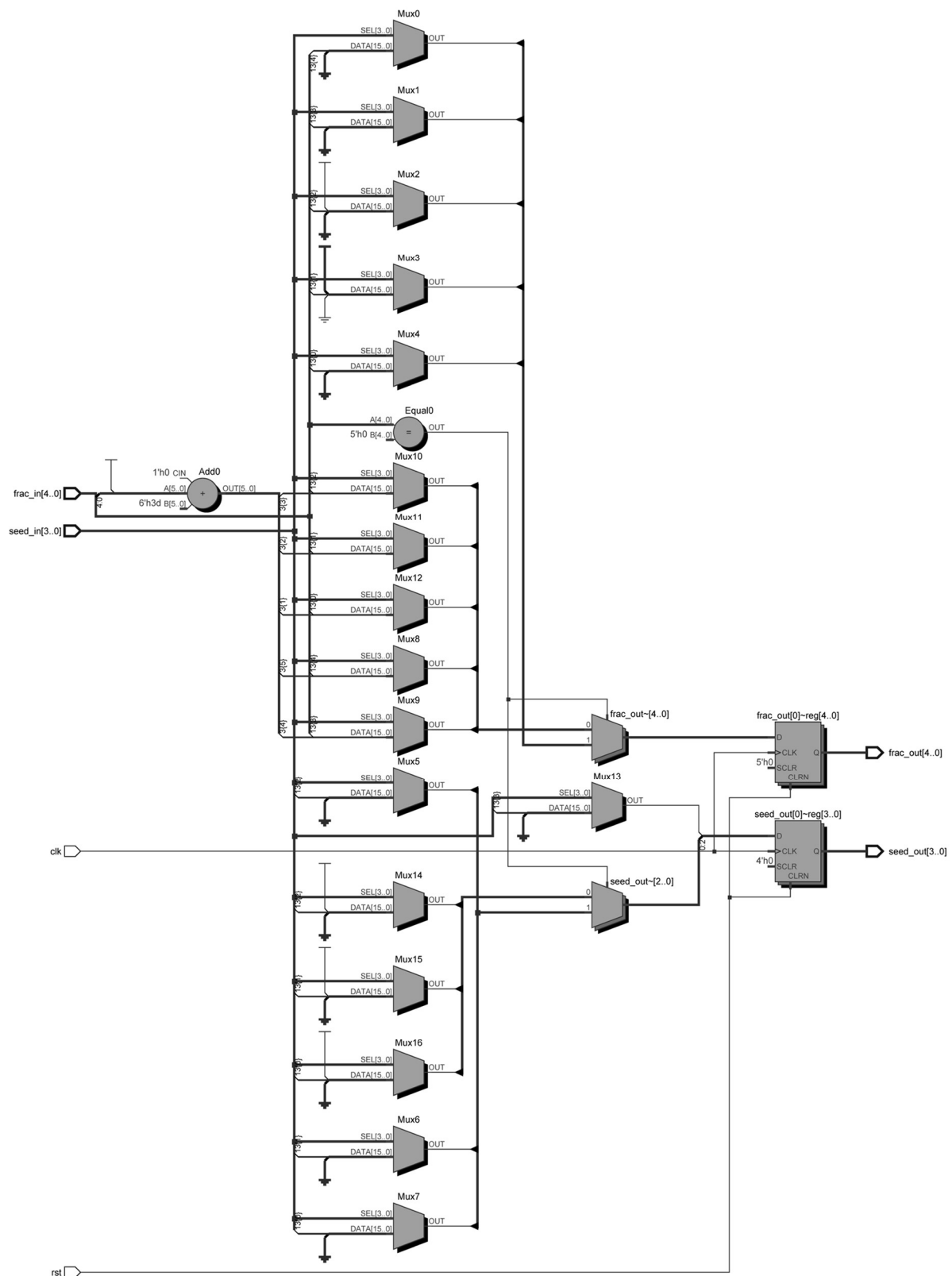


Figure 6. RTL graphical representation of the uniform encoding block.

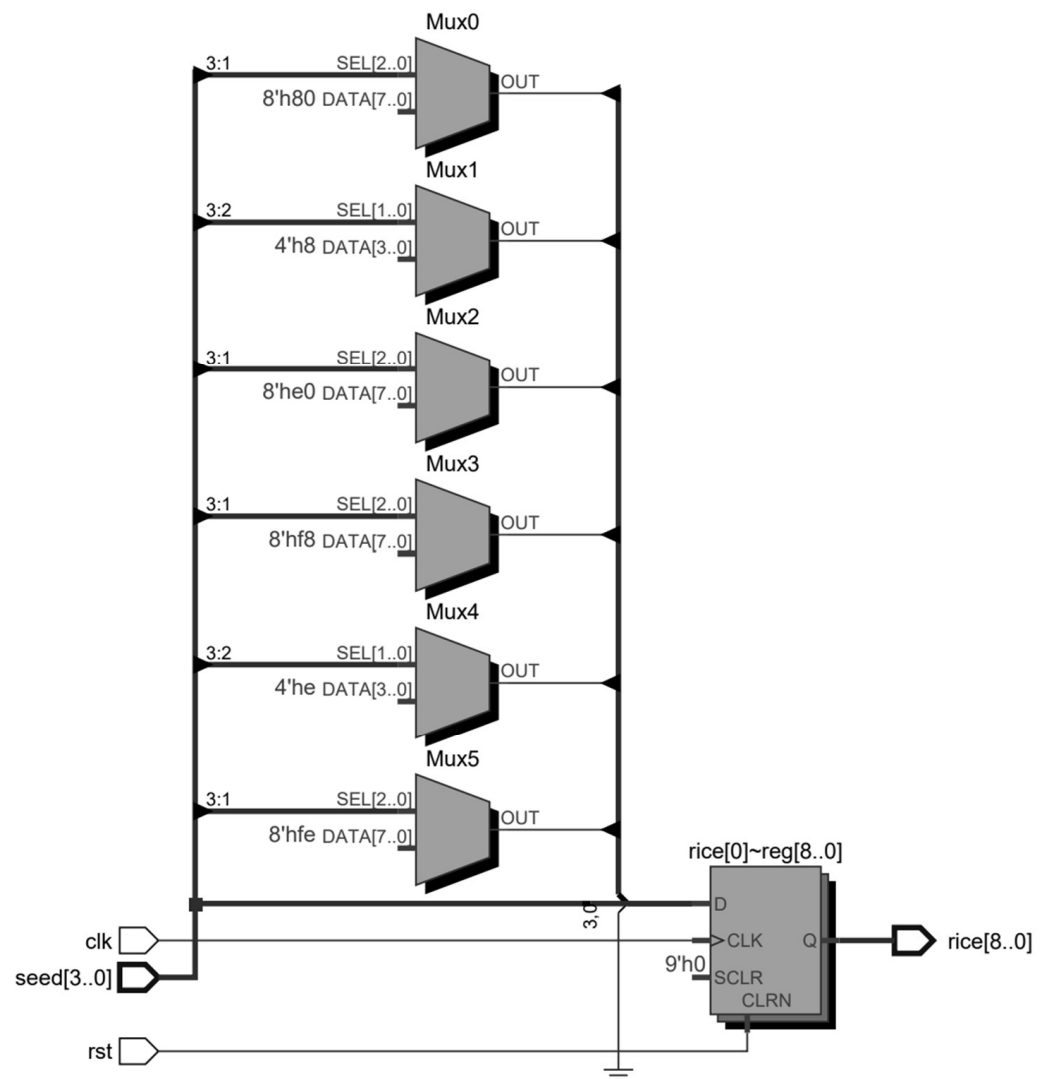


Figure 7. RTL graphical representation of the second step of the direct Rice encoder.

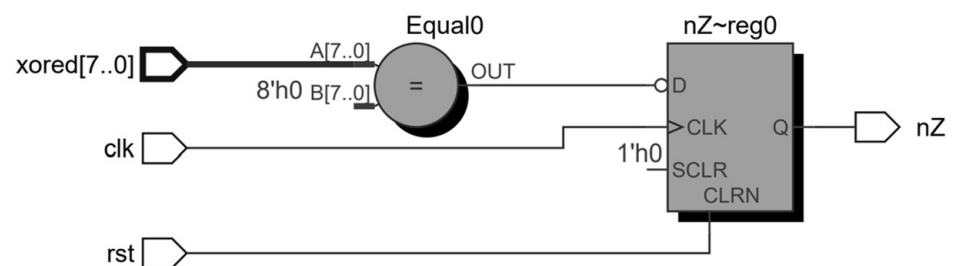


Figure 8. RTL graphical representation of zero detection logic.

3.2.8. Adaptive Run-Length Encoder

The run-length encoder constructs the initial structure of the encoded stream, comprising two vectors: one vector records the count of bits for each run, and the other vector captures the number of bits needed to represent this count. To optimize the processing time and keep the clock period short, the encoding process is divided into two distinct steps, each handling one of the vectors. In the first step, the encoder utilizes a 32-bit counter to track consecutive bits. The counter width is large enough to accommodate more than 21 of the largest scenes in the dataset, ErtaAle of Hyperion. During each clock cycle, the component compares the current input bit with the previous one. If they match, it increments

the counter. When the input changes, it updates the output count with the current counter value and resets the counter for the new run. Finally, when the signal “done” is asserted, it captures the last count value. Figure 9 exhibits the RTL representation of the initial stage of the run-length encoder.

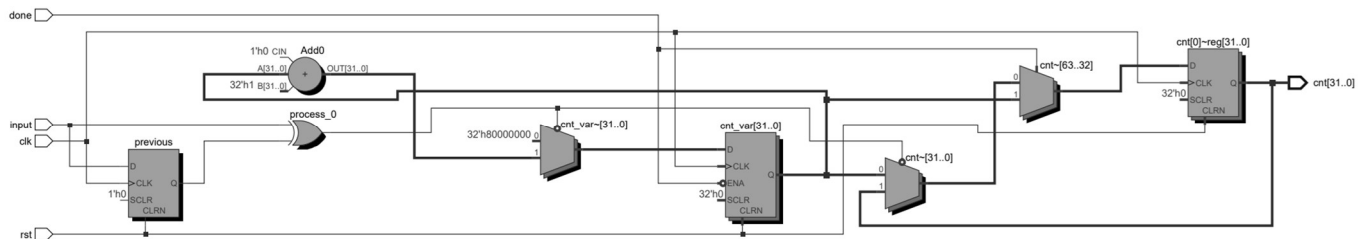


Figure 9. RTL graphical representation of the first stage of the run-length encoder.

In the second stage of the run-length encoder, the objective is to determine the minimum number of bits, denoted as n , based on the value derived from a 32-bit counter produced in the initial phase. This process employs a binary search algorithm, optimized for efficiency. During each clock cycle, the component examines the counter value using a decision tree to deduce the bit length of the input counter. By comparing the counter value against predefined thresholds (pivots), it allocates n reflecting the length of the run. While the binary search for the range of a 32-bit number may involve complex nested conditional statements, the resulting RTL representation simplifies this complexity to just five logic levels, as shown in Figure 10.

3.2.9. The Glue Logic

The glue logic serves as the coordinating unit for the previous components to achieve an efficient compression algorithm. It starts by receiving an 8-bit unsigned input, which is first processed by the XOR logic to reduce the entropy of the input data. The zero-detection logic monitors the XORed output to set a flag, in case the input is non-zero, which in turn aids in subsequent data handling. The resulting XORed value is used by the shift amount calculator to determine the appropriate shift amount based on the minimum number of bits required to represent the XORed values. Given the precalculated shift amount, these XORed values are passed to the seed generation component to compute the seed value. Together with a delayed output of the XORed data, the seed is utilized by the fraction calculation logic to compute the fractional part of the estimated square root. Finally, the uniform encoding block encodes both the seed and fractional parts to maintain a consistent encoding scheme. Next, we present in Figure 11 a schematic diagram illustrating how these components are interconnected within the system.

3.2.10. Top-Level Entity

The top-level entity is designed to process multiple streams of data simultaneously using a specialized lossless compressor. It comprises multiple instances of this compressor, each dedicated to processing its respective data stream independently and simultaneously. This parallel processing architecture is facilitated by replicating the interconnected components, with the number of replicas constrained by the general-purpose I/O (GPIO) pins of the device. This design aims to efficiently and rapidly handle large volumes of data, such as hyperspectral images, in real time, leading to the optimization of both throughput and power requirement.

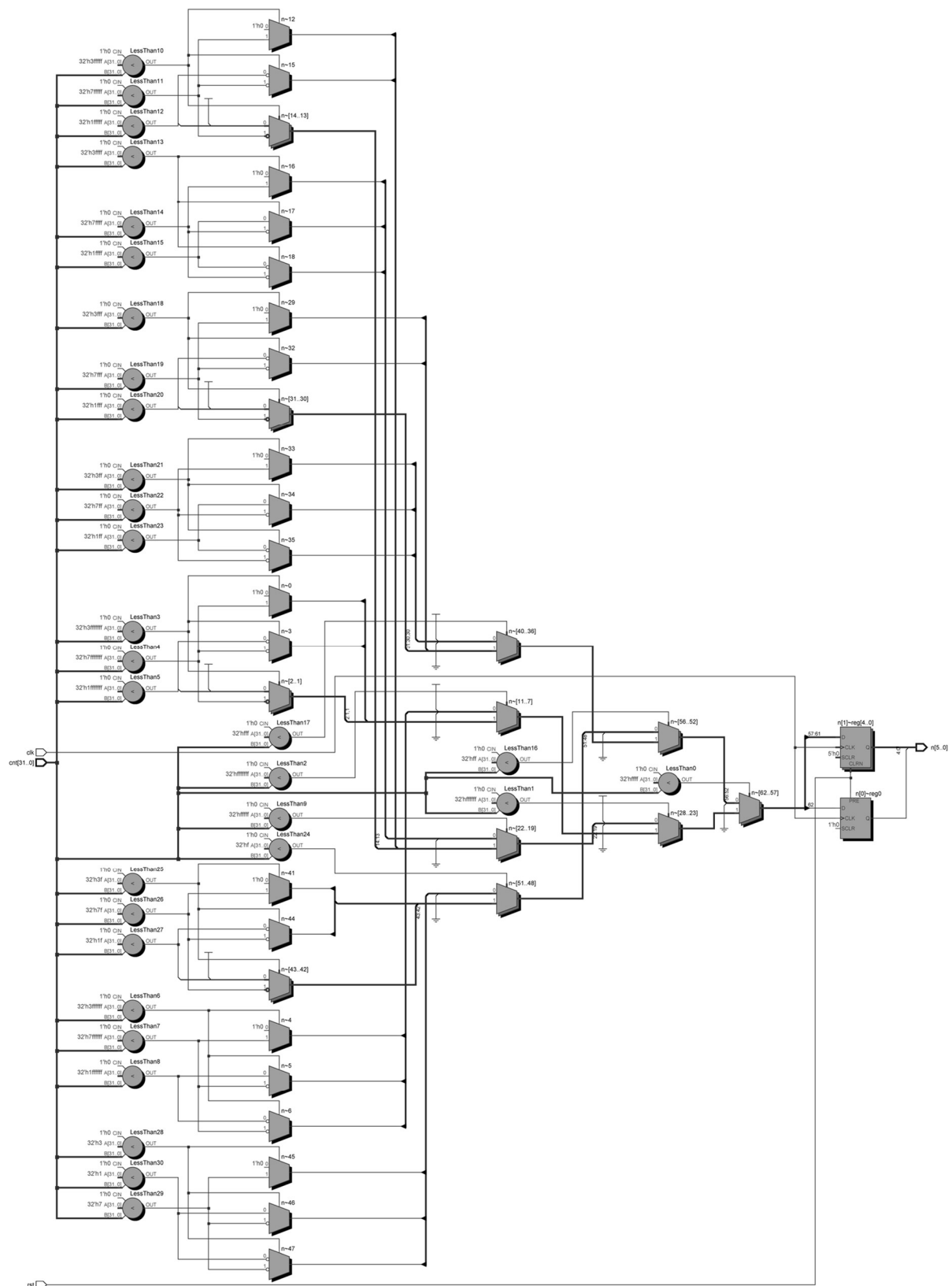


Figure 10. RTL graphical representation of the second step of the run length encoder.

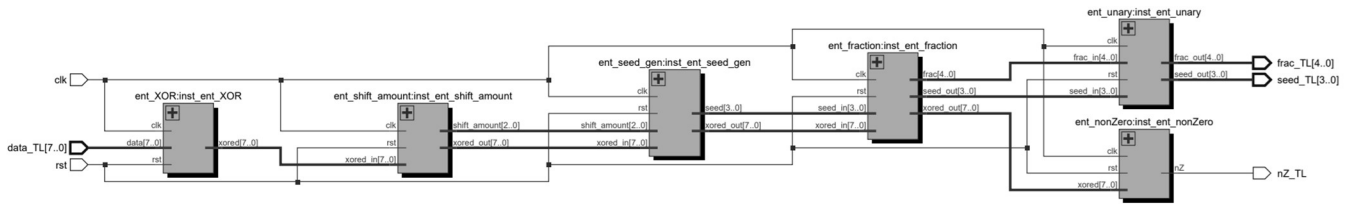


Figure 11. Schematics showing the interconnections between components within the top-level entity of the lossless compressor.

4. Results and Discussion

In this study, we focus primarily on the hardware implementation aspects of the compression algorithm, emphasizing metrics such as clock frequency, power consumption, resource utilization, throughput, and scalability. Other metrics at the algorithmic level pertaining to computational complexity, compression ratio, accuracy, and fidelity of the decompressed data are presented in [20]. For the analysis of our FPGA-based compression, we utilized Quartus Prime for FPGA programming and timing analysis, while ModelSim was employed for comprehensive simulations. VHDL was selected as the design language to ensure precise control over the hardware. Performance testing and simulations were conducted on a system equipped with an Intel(R) Core(TM) i7-10510U CPU, clocked at 1.80 GHz (up to 2.30 GHz), with 16 GB of RAM, and running on a Windows 11 Operating System. Our implementation targeted the Intel Cyclone V GT FPGA (model 5CGTFD9E5F35C7) for its number of GPIO pins. This model is engineered to deliver high performance and low power consumption, making it suitable for a wide range of applications. Table 4 reveals some of the key design specifications of this FPGA model.

Table 4. Key design characteristics of the Intel Cyclone V GT FPGA (model 5CGTFD9E5F35C7) [29].

FPGA Characteristics	Name/Value
Manufacturer	Intel
Series	Cyclone® V GT
Number of LABs/CLBs	113560
Number of Registers	4786
Number of DSP Blocks	342
Total RAM Bits	14251008
Number of Pins	616
Number of GPIO Pins	560
Voltage—Supply	1.07 V~1.13 V
Operating Temperature	0 °C~85 °C

4.1. Clock Frequency

In the timing analysis performed using Quartus Prime, the hardware was carefully designed to complete one cycle within 10 nanoseconds. This means, all signals propagate through the components and complete their operations within the specified period. This setup resulted in a positive slack, thus ensuring the circuit functions correctly and synchronously with the clock signal while maintaining the required timing constraints for reliable performance. Consequently, the system achieved a maximum operating frequency of 103.14 MHz, indicating a well-optimized circuit layout.

The maximum operating frequency at which the circuit can operate without timing issues is 103.14 MHz. Evidently, a 10-nanosecond clock period corresponds to a frequency of 100 MHz. However, the system achieved a slightly higher frequency of 103.14 MHz, indicating an optimized performance.

4.2. Power Requirement

Multiple optimization techniques are examined during the compilation process. When the compiler is set to prioritize the “Power” or “Balanced” modes, the resulting power

consumption of the FPGA design is 0.874 Watts, indicating a lower power usage. On the other hand, when the optimization focus is shifted to prioritize “Speed”, the power consumption of the design increases to 1.018 Watts. Despite the different optimization focuses, the maximum operating frequency of the circuit remains nearly unchanged. This suggests a well-balanced design that effectively utilizes the FPGA capabilities, reaching near-optimal performance limits set by the hardware physical constraints.

4.3. Throughput

The Cyclone V FPGA has 560 GPIO pins that allow the FPGA to interface with other devices and handle various input and output signals. The available number of pins allows for the accommodation of 31 independent units, enabling the system to process multiple data streams simultaneously. This level of parallelism increases the system’s overall performance enabling high data throughput and fast processing speeds.

Each of the 31 units processes half a sample, equal to 8 bits, per clock cycle, given that one pixel has a resolution of 16 bits. Since there are 31 units, the total number of 16-bit samples processed per cycle across all units is thus equal to 15.5. This is evident in the following formulae:

$$\text{Total samples per cycle} = \frac{31 \text{ units} \times 8 \text{ bits}}{16 \text{ bits per sample}} = 15.5 \text{ samples.}$$

The overall throughput of the system is calculated by multiplying the number of samples processed per cycle by the maximum operating frequency of the FPGA. To find the throughput in Mega Samples per second, we multiply the samples per cycle by the maximum frequency achieved, as follows:

$$\text{Throughput} = 15.5 \text{ samples per cycle} \times 103.14 \text{ MHz} = 1598.67 \text{ MSps}$$

This means that, with an operating frequency of 103.14 MHz (or 103.14 Mega cycles per second), the system achieves a notable throughput of 1598.67 MSps.

4.4. Resource Utilization and Scalability

Results obtained by the compilation report show less than 1% utilization of logic elements. This suggests that the design is not heavily reliant on general logic processing. On the other hand, results also show a 18% utilization of the DSP blocks. These blocks are specialized components used for tasks like filtering, multiplication, and other signal processing operations. Using 18% of these blocks indicates a moderate level of signal processing activity. Memory blocks are used for storing data temporarily and the fact that none are used indicates that the design processes data in real time without needing to store it temporarily. There is significant room for enhancement since only a small fraction of the FPGA’s resources are used. It follows that there is plenty of capacity left for adding more features or improving existing ones. However, the full utilization of all 560 I/O pins creates a bottleneck, limiting expansion related to external connections. Any scalability efforts would need to address this bottleneck, possibly through multiplexing techniques, or upgrading to a larger FPGA. Multiplexing is a method that allows multiple signals to share a single I/O pin. This could help overcome the potential bottleneck by effectively increasing the number of connections without needing more pins [30].

4.5. Comparison with State-of-the-Art Implementations

In this section, we evaluate our hardware-accelerated compression algorithm by comparing it with select studies from the related work that provide detailed metrics, specifically focusing on throughput and power requirement. Our selection is based on the availability of comprehensive data essential for a meaningful comparison. This analysis extends the initial findings from our earlier systematic review, enhancing our understanding of the algorithm performance in these key areas. Table 5 provides a detailed comparison

of our hardware-accelerated compression algorithm against other related state-of-the-art algorithms. We note here that efficiency values are calculated by dividing the throughput (MSps) by power requirements (Watts).

Table 5. Performance results of the hardware acceleration of related lossless compression algorithms for HSIs and their comparison with our proposed implementation of our lossless compressor.

Compression Method	CR	Throughput (MSps)	Power (Watts)	Efficiency (MSps/W)	Reference
DPCM	4.8	280	650	0.4	[31]
CCSDS123	2.2–4.5	183.4	60	3.1	[32]
CCSDS 123	-	401–116	60–15	6.7–7.7	[33]
CCSDS123	3.2–4	165.65	2.6	63.7	[34]
CCSDS 123	1.5–5.5	129	4.9	26.3	[35]
CCSDS123	-	69.8	4.56	15.3	[36]
CCSDS123	-	93.2	4.56	20.4	[36]
CCSDS123	-	45	5.7	7.9	[37]
CCSDS123	-	146.9	6.28	23.4	[37]
CCSDS123	-	308.13	10.9	28.3	[37]
CCSDS123	-	66	5.7	11.6	[37]
CCSDS123	-	203.3	6.28	32.3	[37]
CCSDS123	-	402.5	10.9	36.9	[37]
CCSDS 123	-	147	0.295	498.3	[38]
CCSDS 123	-	750	0.515	1456	[39]
CCSDS123	3.4	3.5	0.169	20.7	[40]
CCSDS123	3.4	11.3	2.345	4.8	[40]
CCSDS123	3.4	11.2	2.345	4.8	[40]
CCSDS 123	2.3	3.5	0.169	20.7	[40]
CCSDS 123	2.3	11.3	2.345	4.8	[40]
CCSDS 123	2.3	11.2	2.345	4.8	[40]
Prediction-based	-	179.7	3.04	59.1	[41]
Prediction-based	-	116	0.95	122.1	[41]
Prediction-based	-	219.4	5.3	41.4	[41]
Prediction-based	-	62.2	65	0.96	[41]
Prediction-based	-	62.6	29	2.2	[41]
CCSDS 123	-	179.7	3.04	59.1	[41]
CCSDS 123	-	116	0.95	122.1	[41]
CCSDS 123	-	219.4	5.3	41.3	[41]
CCSDS 123	-	62.2	65	0.96	[41]
CCSDS 123	-	62.6	29	2.2	[41]
VS—3DGAP—ExtRice (CCSDS based)	2.8	210	0.573	366.5	[42]
Prediction-based	2.5	23.3	0.55	42.4	[43]
CCSDS 123	2.5	23.3	0.55	42.4	[43]
CCSDS 123.0-B-3	-	9.38	0.149	63	[16]
CCSDS 123.0-B-2	-	249.6	1.21	206.3	[17]
CCSDS-123.0-B-2	-	305	1.525	200	[18]
Proposed	2.6	1598.67	0.874	1829.1	-

The table above provides a detailed comparison of several compression algorithms, with varying metrics that signify their efficiency and practicality in different scenarios. Among these, the study presented in [35] exhibits the highest compression ratio, which reaches up to 5.5, indicating its effectiveness in reducing data size. Conversely, the method with the lowest power consumption is found in [16], consuming just 0.149 Watts, demonstrating its suitability for energy-critical applications. Our proposed method, while not leading in these specific categories, is noted for its high throughput (1598.67 MSps) and efficiency (1829.1 MSps/W), indicating its potential utility in scenarios that require high performance and energy efficiency.

5. Conclusions

In this paper, we have presented an analysis and implementation of the hardware acceleration of a novel lossless compression algorithm tailored for hyperspectral data. This algorithm is designed for real-time processing in satellite systems and utilizes an innovative seed generation technique for square root calculations, optimized for modern hardware architectures, including FPGA platforms.

Our comparative analysis with other state-of-the-art hardware-accelerated compression algorithms reveals that, while our system does not achieve the highest compression ratio, it excels in operational throughput and energy efficiency. It achieves a notable throughput of 1598.67 MSps and maintains a low power consumption of under 1 Watt, resulting in an efficiency rate of 1829.1 MSps/Watt. These characteristics make our algorithm particularly well suited for environments where high performance and energy efficiency are crucial.

6. Future Work

Despite the significant improvements in throughput and power consumption, there are opportunities to enhance scalability and throughput by addressing the bottleneck caused by the full utilization of I/O pins. Future work could focus on refining these compression methods to improve performance further. Enhancing compression ratios would be particularly effective for environments with strict storage and bandwidth limitations. Additionally, implementing these compression techniques across different FPGA platforms could provide valuable insights into performance and energy consumption variations. Furthermore, integrating security features such as encryption into the compression process would protect sensitive data in critical applications. There is also potential to integrate machine learning algorithms into the compression process, introducing dynamic, intelligent adjustments to optimize trade-offs between compression ratio, processing speed, and data fidelity. These advancements would broaden the scope of this research and enhance its practical applications, paving the way for continuous technological evolution.

Author Contributions: Conceptualization, A.A. and B.B.Y.; methodology, A.A. and B.B.Y.; investigation, A.A. and B.B.Y.; writing—original draft preparation, A.A.; writing—review and editing, B.B.Y.; supervision, B.B.Y. All authors have read and agreed to the published version of the manuscript.

Funding: This research received no external funding.

Data Availability Statement: The original contributions presented in the study are included in the article, further inquiries can be directed to the corresponding author.

Acknowledgments: The authors would like to gratefully acknowledge the support of the Deanship of Scientific Research at King Saud University (KSU), Riyadh, Saudi Arabia.

Conflicts of Interest: The authors declare no conflicts of interest.

References

1. Liu, C.; Xing, C.; Hu, Q.; Wang, S.; Zhao, S.; Gao, M. Stereoscopic hyperspectral remote sensing of the atmospheric environment: Innovation and prospects. *Earth-Sci. Rev.* **2022**, *226*, 103958. [CrossRef]
2. Flores, H.; Lorenz, S.; Jackisch, R.; Tusa, L.; Contreras, I.C.; Zimmermann, R.; Gloaguen, R. UAS-Based Hyperspectral Environmental Monitoring of Acid Mine Drainage Affected Waters. *Minerals* **2021**, *11*, 182. [CrossRef]
3. Lu, B.; Dao, P.D.; Liu, J.; He, Y.; Shang, J. Recent Advances of Hyperspectral Imaging Technology and Applications in Agriculture. *Remote Sens.* **2020**, *12*, 2659. [CrossRef]
4. Wang, C.; Liu, B.; Liu, L.; Zhu, Y.; Hou, J.; Liu, P.; Li, X. A review of deep learning used in the hyperspectral image analysis for agriculture. *Artif. Intell. Rev.* **2021**, *54*, 5205–5253. [CrossRef]
5. Ang, K.L.-M.; Seng, J.K.P. Big Data and Machine Learning with Hyperspectral Information in Agriculture. *IEEE Access* **2021**, *9*, 36699–36718. [CrossRef]
6. Samuels, A.C. Military Applications of Portable Spectroscopy. In *Portable Spectroscopy and Spectrometry*; Wiley: West Maitland, FL, USA, 2021; pp. 149–157.
7. Dua, Y.; Kumar, V.; Singh, R.S. Comprehensive review of hyperspectral image compression algorithms. *Opt. Eng.* **2020**, *59*, 090902. [CrossRef]
8. Altamimi, A.; Ben, B. Youssef, A Systematic Review of Hardware-Accelerated Compression of Remotely Sensed Hyperspectral Images. *Sensors* **2021**, *22*, 263. [CrossRef]
9. Caba, J.; Díaz, M.; Barba, J.; Guerra, R.; López, J. FPGA-Based On-Board Hyperspectral Imaging Compression: Benchmarking Performance and Energy Efficiency against GPU Implementations. *Remote Sens.* **2020**, *12*, 3741. [CrossRef]
10. Melián, J.M.; Jiménez, A.; Díaz, M.; Morales, A.; Horstrand, P.; Guerra, R.; López, S.; López, J.F. Real-Time Hyperspectral Data Transmission for UAV-Based Acquisition Platforms. *Remote Sens.* **2021**, *13*, 850. [CrossRef]
11. Calin, M.A.; Calin, A.C.; Nicolae, D.N. Application of airborne and spaceborne hyperspectral imaging techniques for atmospheric research: Past, present, and future. *Appl. Spectrosc. Rev.* **2021**, *56*, 289–323. [CrossRef]
12. Wildenstein, D.; George, A.D. Towards intelligent compression of hyperspectral imagery. In Proceedings of the 2021 IEEE International Conference on Electronics, Computing and Communication Technologies (CONECCT), Bangalore, India, 9–11 July 2021.
13. Zhang, L.; Liu, T.; Zhang, K. Design and Implementation of Lossless Compression System for CCSDS Hyperspectral Images. In *Journal of Physics: Conference Series*; IOP Publishing: Bristol, UK, 2021.
14. CCSDS-123.0-B-1; Lossless Multispectral & Hyperspectral Image Compression. The Consultative Committee for Space Data Systems (CCSDS). 2012. Available online: <https://public.ccsds.org/Pubs/123x0b1ec1s.pdf> (accessed on 20 April 2024).
15. CCSDS-123.0-B-2; Low-Complexity Lossless and Near-Lossless Multispectral and Hyperspectral Image Compression. The Consultative Committee for Space Data Systems (CCSDS). 2019. Available online: <https://public.ccsds.org/Pubs/120x2g2.pdf> (accessed on 20 April 2024).
16. Grignani, W.; Santos, D.A.; Dillillo, L.; Viel, F.; Melo, D.R. A Low-Cost Hardware Accelerator for CCSDS 123 Lossless Hyperspectral Image Compression. In Proceedings of the 2023 IEEE International Symposium on Defect and Fault Tolerance in VLSI and Nanotechnology Systems (DFT), Juan-Les-Pins, France, 3–5 October 2023.
17. Báscones, D.; Gonzalez, C.; Mozos, D. A real-time FPGA implementation of the ccstds 123.0-b-2 standard. *IEEE Trans. Geosci. Remote Sens.* **2022**, *60*, 1–13. [CrossRef]
18. Chatziantoniou, P.; Tsigkanos, A.; Theodoropoulos, D.; Kranitis, N.; Paschalis, A. An Efficient Architecture and High-Throughput Implementation of CCSDS-123.0-B-2 Hybrid Entropy Coder Targeting Space-Grade SRAM FPGA Technology. *IEEE Trans. Aerosp. Electron. Syst.* **2022**, *58*, 5470–5482. [CrossRef]
19. Shaharim, N.A.N.; Chez, T.L.; Zainul, A.Z.; Noor, N.R.M. Parallelization of CCSDS Hyperspectral Image Compression Using OpenMP. *J. Eng. Sci.* **2022**, *18*, 1–16.
20. Altamimi, A.; Ben Youssef, B. Lossless and Near-Lossless Compression Algorithms for Remotely Sensed Hyperspectral Images. *Entropy* **2024**, *26*, 316. [CrossRef]
21. Putra, R.V.W. A novel fixed-point square root algorithm and its digital hardware design. In Proceedings of the International Conference on ICT for Smart Society, Jakarta, Indonesia, 13–14 June 2013.
22. Altamimi, A.; Ben Youssef, B. Novel seed generation and quadrature-based square rooting algorithms. *Sci. Rep.* **2022**, *12*, 20540. [CrossRef]
23. Fog, A. Instruction tables: Lists of instruction latencies, throughputs and micro-operation breakdowns for Intel, AMD and VIA CPUs. *Cph. Univ. Coll. Eng.* **2011**, *93*, 110.
24. Blinn, J. *Jim Blinn's Corner: Notation, Notation, Notation*; Morgan Kaufmann: Burlington, MA, USA, 2003.
25. Cooper, T.K. Exclusive-or Preprocessing and Dictionary Coding of Continuous-Tone Images. Ph.D. Dissertation, University of Louisville, Louisville, KY, USA, 2015.
26. Mansour, A.M.; El-Sawy, A.M.; Aziz, M.S.; Sayed, A.T. A New Hardware Implementation of Base 2 Logarithm for FPGA. *Int. J. Signal Process. Syst.* **2015**, *3*, 171–181. [CrossRef]
27. Warren, H.S. *Hacker's Delight*; Pearson Education: London, UK, 2012.
28. Mehlhorn, K.; Sanders, P. Algorithms and Data structures. In *the Basic Toolbox*; Springer: Berlin/Heidelberg, Germany, 2007; p. 295.

29. DigiKey. Cyclone V FPGA 5CGTDF9E5F35C7 Specifications. Available online: <https://www.digikey.com/en/products/detail/intel/5CGTDF9E5F35C7N/3879603> (accessed on 20 May 2024).
30. Zong, Z. Pin multiplexing optimization in FPGA prototyping system. In Proceedings of the 2017 4th International Conference on Systems and Informatics (ICSAI), Hangzhou, China, 11–13 November 2017.
31. Aranki, N.; Bakhshi, A.; Keymeulen, D.; Klimesh, M. Fast and adaptive lossless on-board hyperspectral data compression system for space applications. In Proceedings of the 2009 IEEE Aerospace Conference, Big Sky, MT, USA, 7–14 March 2009.
32. Hihara, H.; Yoshida, J.; Ishida, J.; Takada, J.; Senda, Y.; Suzuki, M.; Seki, T.; Ichikawa, S.; Ohgi, N. Fast compression implementation for hyperspectral sensor. In *Multispectral, Hyperspectral, and Ultraspectral Remote Sensing Technology, Techniques, and Applications III*; SPIE: Washington, DC, USA, 2010.
33. Nambu, T.; Takada, J.; Kawashima, T.; Hihara, H.; Inada, H.; Suzuki, M.; Seki, T.; Ichikawa, S. Development of onboard fast lossless compressors for multi and hyperspectral sensors. In *Multispectral, Hyperspectral, and Ultraspectral Remote Sensing Technology, Techniques and Applications IV*; SPIE: Washington, DC, USA, 2012.
34. Hwang, Y.-T.; Lin, C.-C.; Hung, R.-T. Lossless Hyperspectral Image Compression System-Based on HW/SW Codesign. *IEEE Embed. Syst. Lett.* **2010**, *3*, 20–23. [[CrossRef](#)]
35. Mamatha, A.; Singh, V. Lossless hyperspectral image compression using intraband and interband predictors. In Proceedings of the 2014 International Conference on Advances in Computing, Communications and Informatics (ICACCI), Delhi, India, 24–27 September 2014.
36. Santos, L.; Vitulli, R.; López, J.F.; Sarmiento, R. GPU implementation of a lossy compression algorithm for hyperspectral images. In Proceedings of the 2012 4th Workshop on Hyperspectral Image and Signal Processing: Evolution in Remote Sensing (WHISPERS), Shanghai, China, 4–7 June 2012.
37. Santos, L.; Magli, E.; Vitulli, R.; López, J.F.; Sarmiento, R. Highly-parallel GPU architecture for lossy hyperspectral image compression. *IEEE J. Sel. Top. Appl. Earth Obs. Remote Sens.* **2013**, *6*, 670–681. [[CrossRef](#)]
38. Wu, J.; Kong, W.; Mielikainen, J.; Huang, B. Lossless Compression of Hyperspectral Imagery via Clustered Differential Pulse Code Modulation with Removal of Local Spectral Outliers. *IEEE Signal Process. Lett.* **2015**, *22*, 2194–2198. [[CrossRef](#)]
39. Tsimpouris, E.; Tsakiridis, N.L.; Theocharis, J.B. Using autoencoders to compress soil VNIR–SWIR spectra for more robust prediction of soil properties. *Geoderma* **2021**, *393*, 114967. [[CrossRef](#)]
40. Giordano, R.; Guccione, P. ROI-Based On-Board Compression for Hyperspectral Remote Sensing Images on GPU. *Sensors* **2017**, *17*, 1160. [[CrossRef](#)] [[PubMed](#)]
41. Bernabé, S.; Martín, G.; Nascimento, J.M. Parallel hyperspectral compressive sensing method on GPU. In *High-Performance Computing in Remote Sensing V*; SPIE: Washington, DC, USA, 2015.
42. Eggho, C.; Vladimirova, T. Adaptive hyperspectral image compression using the KLT and integer KLT algorithms. In Proceedings of the 2014 NASA/ESA Conference on Adaptive Hardware and Systems (AHS), Leicester, UK, 14–17 July 2014.
43. Santos, L.; Berrojo, L.; Moreno, J.; Lopez, J.F.; Sarmiento, R. Multispectral and Hyperspectral Lossless Compressor for Space Applications (HyLoC): A Low-Complexity FPGA Implementation of the CCSDS 123 Standard. *IEEE J. Sel. Top. Appl. Earth Obs. Remote Sens.* **2015**, *9*, 757–770. [[CrossRef](#)]

Disclaimer/Publisher’s Note: The statements, opinions and data contained in all publications are solely those of the individual author(s) and contributor(s) and not of MDPI and/or the editor(s). MDPI and/or the editor(s) disclaim responsibility for any injury to people or property resulting from any ideas, methods, instructions or products referred to in the content.