DECOMPRESSION OF SATELLITE TELEMETRY DATA USING CCSDS 121.0-B-3 WITH RICE DECODING

BTP PROJECT REPORT FOR END-SEM EVALUATION

DEPARTMENT OF ELECTRICAL AND ELECTRONICS ENGINEERING

Indian Institute of Technology Patna

Patna, Bihar, 801106

Submitted By:

Girish Sai Krishna Akula

2201EE06

Electrical and Electronics Engineering

Panel Members:

Prof. Sanjoy Kumar Parida (Instructor)

Prof. S.Sivasubramani

ABSTRACT

There is a vast amount of data being continuously collected by satellites, which must be compressed before transmission to ground-based receiver stations due to bandwidth and energy constraints. Consequently, there is a growing need for reliable and standards-compliant **decompression systems** to ensure accurate recovery of scientific and telemetry data. This report presents the design and implementation of a decompressor targeting a subset of the **CCSDS 121.0-B-3 Lossless Data Compression** standard, specifically focusing on **fixed-length (FS)** and **split-sample coding** options with **1D prediction**, as applied to satellite data streams.

A key feature of this decompressor is its support for **Rice codes**, a form of entropy coding widely adopted in space applications due to its simplicity and effectiveness on low-entropy residuals. These Rice codes are decoded using an optimized variablelength decoding strategy that reduces computational complexity while preserving alignment accuracy.

To further enhance decoding performance and address the challenges posed by irregular bitstream alignment in Rice-coded data, the design incorporates a **plane separation technique**, adapted from *A Fast Variable-Length Decoder Using Plane Separation* by Jeon et al. The architecture has been modified to suit Rice decoding in CCSDS-compliant workflows. In this approach, the input stream is split into two independent logical planes, enabling concurrent execution of key functions such as **barrel shifting**, **bit alignment**, and **latch control**. This separation significantly reduces feedback path delays and improves throughput during the decoding of variable-length encoded segments.

The decompression flow was architected after an in-depth study of the CCSDS 121.0B3 standard, with particular emphasis on **packet header parsing**, **reference sample reconstruction**, and **error-to-sample conversion** for both FS and Rice-coded data. A **modular architecture** was adopted, ensuring clean separation between core components such as the input and OR planes, control path, carry generator, and the 1D inverse prediction logic. The prediction module, based on spatial relationships between neighboring samples, was rigorously designed to conform to the standard's specifications.

During implementation, key challenges emerged in maintaining precise bit-level synchronization—especially across **split-sample boundaries** and while managing dynamic code lengths. The adoption of plane separation proved instrumental in mitigating these issues by isolating shift and decodes operations, offering not only correctness but also architectural scalability.

1. INTRODUCTION

1.1. Introduction

Reliable data decompression is a critical requirement in space missions, where the combination of limited bandwidth and constrained onboard processing capabilities necessitates highly efficient and standards-compliant data handling. The CCSDS 121.0B-3 Lossless Data Compression standard provides a structured approach for achieving lossless compression through various encoding methods, including fixedlength, splitsample, and Rice coding. This project focuses on implementing a decompressor compatible with selected components of this standard, specifically fixed-length and split-sample modes using 1D prediction.

The complexity introduced by variable-length Rice coding poses unique challenges, particularly in terms of alignment and decoding efficiency. To address this, the project integrates the plane separation method, a high-throughput decoding architecture proposed by Jeon et al. in "A Fast Variable-Length Decoder Using Plane Separation." This method enables concurrent execution of critical operations within the feedback path, significantly improving throughput without sacrificing accuracy or compliance.

1.2. Project Objective

The goal of this project is to develop a CCSDS 121.0-B-3 compliant decompression system that efficiently supports fixed-length and split-sample coding modes while incorporating variable-length Rice code decoding. By leveraging the plane separation method, the system aims to reduce decoding latency and improve performance. The design emphasizes a modular and lightweight architecture to ensure suitability for implementation in resource-constrained space systems while maintaining strict compliance with CCSDS specifications.

1.3. Technical Approach

The architecture of the decompressor consists of several interrelated modules designed to process CCSDS-compressed data efficiently. A bitstream parser extracts control headers and encoded values from the input stream. The Rice decoder, based on the plane separation method, employs two independent planes to manage input data alignment and decision logic concurrently. This dual-plane configuration ensures that the most time-sensitive operations in Rice decoding are parallelized, thereby improving throughput.

The prediction engine implements one-dimensional prediction using neighbouring sample values, aligning with the CCSDS standard's reconstruction logic. The error mapping reverser translates the decoded prediction errors back into original

sample values, while the output generator assembles these into complete decompressed blocks. Together, these modules enable accurate reconstruction of data with minimal latency.

1.4. Significance of the Project

This project demonstrates a focused and efficient approach to implementing a decompressor tailored for specific modes of the CCSDS 121.0-B-3 standard. By incorporating the plane separation technique into the decoding of Rice-coded data, it presents a novel solution that addresses both performance and compliance. The resulting system is not only lightweight and modular but also capable of real-time decompression, which is essential for onboard processing in satellite missions.

2. AN OVERVIEW OF ENTROPY CODING

The CCSDS 121.0-B-3 standard employs Rice coding as a central method for encoding prediction errors derived from lossless 1D prediction models. Rice coding is chosen for its simplicity, computational efficiency, and suitability for representing small, nonnegative integers—which are typical of prediction residuals in telemetry and image data. The standard supports two main encoding structures that utilize Rice coding either directly or indirectly: the *fixed-length* and *split-sample* coding options.

2.1.Fixed-Length Coding Option

In the fixed-length coding mode, prediction errors are first mapped to non-negative integers through a predefined transformation and then encoded using a constant bit width, referred to as the sample field length. Although Rice coding is not explicitly applied as a variable-length code in this mode, the mapping stage aligns with the Rice framework where a fixed parameter (in this case, the field length) is applied uniformly across all samples.

This mode provides highly efficient decoding since each encoded value occupies a predictable and uniform number of bits. It is best suited for blocks where the prediction error range is narrow and consistent, allowing for the optimal selection of a single field length that minimizes redundancy while avoiding overflow. The simplicity of this scheme makes it well-suited for implementation in environments with limited processing power and strict real-time constraints.

Preprocessed Sample Values, δ_i	FS Codeword
0	1
1	01
2	001
.	.
.	.
.	.
2^n-1	0000 ... 00001 (2^n-1 zeros)

Fig 2.1: Fundamental Sequence Codewords as a Function of the Preprocessed Samples

2.2. Split-Sample Coding Option

The split-sample coding mode introduces adaptive compression by applying Rice coding in a variable-length format to alternating subsets of samples—specifically, even- and odd-indexed samples within a block. Each subset is encoded independently, with its own Rice parameter determined by the encoder based on the local statistics of the prediction errors. This interleaving strategy allows the compression algorithm to exploit localized redundancy more effectively, improving compression efficiency for data with varying statistical properties.

During decoding, the header information is first parsed to retrieve the Rice parameters for each subset. The decoder then applies variable-length decoding to each segment, reconstructs the error sequence through interleaving, and finally performs inverse

prediction to recover the original samples. This mode demands more sophisticated bitstream management due to the non-uniform code lengths, making techniques such as plane separation particularly beneficial for maintaining decoding throughput and bit alignment.

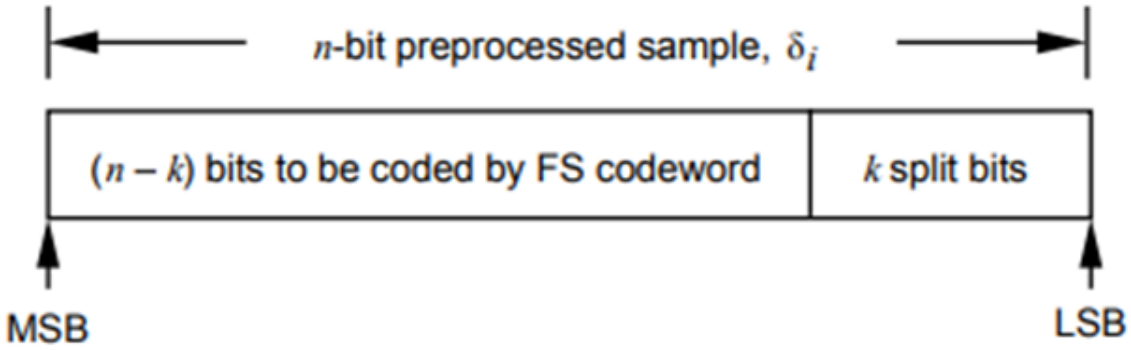

Fig 2.2: Split-Sample Format

Both fixed-length and split-sample coding options leverage the strengths of Rice coding directly or structurally to achieve efficient, lossless compression in space data systems. Their integration into the CCSDS 121.0-B-3 standard reflects a balance between performance, simplicity, and adaptability for mission-specific requirements.

3. CDS PACKET FORMATS

CDSes are packaged into packets, groups of packets, or files. The requisite packet or file formats allow provision of parameters required in order to transfer the adaptive variable-length losslessly coded data between the coder and the telemetry channel packet formatter, as well as compressor parameters needed for recovering the original data that do not change with every CDS. An Option Identification (ID) Key is included at the beginning of every CDS to indicate to the decoder the encoding option used for the corresponding data block. Additionally, the detailed formatting of a CDS depends on whether or not the corresponding data block includes a reference sample.

Code Option	Resolution					
	Basic:	–	–	$n \leq 8$	$8 < n \leq 16$	$16 < n \leq 32$
	Restricted:	$n = 1, 2$	$n = 3, 4$	$4 < n \leq 8$	$8 < n \leq 16$	$16 < n \leq 32$
Zero-Block		00	000	0000	00000	000000
Second-Extension		01	001	0001	00001	000001
FS		—	01	001	0001	00001
k=1		—	10	010	0010	00010
k=2		—	—	011	0011	00011
k=3		—	—	100	0100	00100
k=4		—	—	101	0101	00101
k=5		—	—	110	0110	00110
k=6		—	—	—	0111	00111
k=7		—	—	—	1000	01000
k=8		—	—	—	1001	01001
k=9		—	—	—	1010	01010
k=10		—	—	—	1011	01011
k=11		—	—	—	1100	01100
k=12		—	—	—	1101	01101
k=13		—	—	—	1110	01110
k=14		—	—	—	—	01111
k=15		—	—	—	—	10000
⋮		⋮	⋮	⋮	⋮	⋮
k=29		—	—	—	—	11110
No-compression		1	11	111	1111	11111
NOTE – '—' indicates no applicable value						

Fig 3.1: Selected Code Option ID

3.1. CDS FORMAT FOR FS AND SPLIT-SAMPLE OPTIONS

The CDS format when a Split-Sample option is selected is shown in figure 3.1. Figure 3.1a) shows the case in which there is a reference sample; figure 3.1b) shows the format when no reference sample is present. The CDS has the following structure when a split-sample option is selected: 1) ID bit sequence optionally followed by an nbit reference sample, 2) compressed data, and 3) concatenated k least-significant bits from each sample. This specification includes the FS option, which is a special case of the Split-Sample option with k=0.

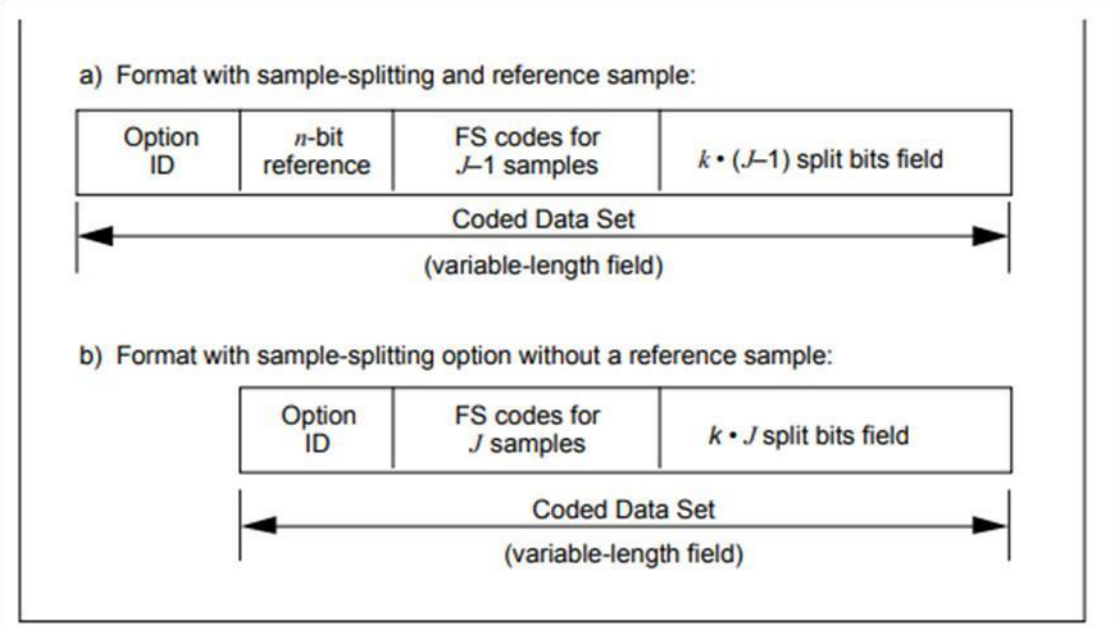

Fig 3.1.1: CDS Format When a Split-Sample Option Is Selected (Including the Special Case k=0 Equivalent to the FS Option)

4. DECODER

4.1. Ideology

To efficiently decode variable-length Rice-coded data within the CCSDS 121.0-B-3 framework, this project implements a hardware-friendly decoding architecture based on plane separation, as introduced by Jeon et al. While traditional variable-length decoders often employ lookup tables (LUTs) for pattern matching, this approach has notable limitations when applied to Rice codes. Specifically, Rice codes are defined by a unary prefix and a fixed-length binary suffix, making their codewords inherently regular and parameter-dependent rather than requiring full symbolic mapping.

Instead of using a LUT-based decoder, this implementation replaces the tablematching logic with a priority encoder, which enables direct detection of the first ‘one’ bit in the unary portion of the codeword. The priority encoder scans the incoming bitstream to determine the length of the unary prefix, which corresponds to the

quotient in Rice coding. This length, combined with the subsequent fixed number of bits (based on the Rice parameter k), fully defines the original encoded value. This method drastically simplifies the decoding logic and reduces memory requirements, especially since the pattern of Rice codes does not necessitate a large or static codebook.

The plane separation method further enhances this process by allowing concurrent execution of bitstream alignment, quotient detection, and decision logic. The bitstream is processed across two planes, an input plane and an OR plane each equipped with barrel shifters and latches. While one plane shifts in new data, the other handles the decoding of the current symbol, ensuring that the carry-out condition and symbol extraction can occur in parallel. This parallelism not only reduces decoding latency but also simplifies handling bit alignment across word boundaries, a common challenge in variable-length coding.

By combining the structural regularity of Rice codes with the efficiency of a hardware-optimized priority encoder and the concurrency offered by plane separation, the resulting decoder achieves high throughput and deterministic performance. This approach is especially beneficial in resource-constrained environments, such as onboard spacecraft systems, where predictable execution and low hardware overhead are critical.

4.2. Decoding Technique

4.2.1. Introduction

Rice coding is a variable-length entropy coding technique that has proven particularly effective in space borne data compression applications. In the CCSDS 121.0-B-3 Lossless Data Compression standard, Rice coding is employed to encode prediction errors—values that typically exhibit an exponentially decaying distribution due to prior transformation via a 1D predictor. Rice codes are simple to encode and decode, require minimal computational resources, and are especially advantageous in real-time or hardware-constrained systems.

This section presents the Rice decoding architecture implemented as part of the CCSDS decompression system described in this project. The decoding logic adheres strictly to the CCSDS 121.0-B-3 specification while introducing significant architectural optimizations for performance. Chief among these is the use of a priority encoder—to replace conventional lookup tables and the incorporation of a plane separation method that supports concurrent decoding and bitstream alignment.

4.2.2. Priority Encoder for Prefix Detection

In traditional Rice decoders, the unary prefix are detected by sequentially scanning the incoming bitstream until a '1' is encountered, indicating the end of the unary portion and the boundary between the quotient and the remainder. This method, although straightforward, introduces sequential latency and is unsuitable for high-speed or hardware-constrained applications, especially when decoding must occur at bit-level granularity in real time. Lookup tables (LUTs) have also been used in past designs to match unary patterns, but they present significant drawbacks in scalability, resource utilization, and logic complexity—particularly when supporting a wide range of Rice parameters or large code lengths.

To overcome these limitations, this project adopts a 64-bit priority encoder as the core mechanism for unary prefix detection. A priority encoder is a combinational logic block that identifies the position of the first occurrence of a specific bit pattern—in this case, the first '1' in a stream of '0's. This position directly corresponds to the quotient q in Rice coding. By scanning an entire 64-bit word in a single clock cycle, the priority encoder outputs the index of the first one without the need for iteration, lookup, or memory access.

This implementation significantly reduces decoding latency because it decouples prefix detection from iterative control flow. The priority encoder provides a deterministic output that allows the decoder to immediately skip the unary prefix and delimiter and jump to the next unary code (if J>1). This result not only improves the throughput but also simplifies control logic, making the design highly compatible with hardware description languages (HDLs) like Verilog or VHDL and easily synthesizable on FPGAs or ASICs. Furthermore, since the priority encoder only uses standard logic gates, its area overhead is minimal compared to LUT-based designs, and it scales efficiently across word sizes. As a result, the use of a priority encoder introduces both performance and structural advantages, making it a highly suitable choice for decoding Rice codes within the CCSDS 121.0-B-3 framework.

4.2.3. Plane Separation Architecture

To support continuous, high-throughput decoding in a streaming bit environment, the decoder uses a *'plane separation architecture'*. The purpose of this architecture is to manage misalignment in the compressed bitstream and enable simultaneous execution of input alignment, symbol decoding, and stream advancement. The bit-aligned nature of Rice codes means that codewords may start or end at arbitrary bit positions across 64-bit word boundaries, and decoding must therefore handle partial symbols with precise bit control.

The decoder separates the processing path into two independent logical planes:

the **input plane** and the **OR plane**. The input plane is responsible for receiving and aligning raw 32-bit input words. Each incoming word is passed through a barrel shifter, which rotates the data to align the first undecoded bit at the beginning of the buffer. This shifted word is then latched and made available for merging. The OR plane retains residual bits from previous decoding cycles and combines them with the newly aligned input using a bitwise OR operation. This OR result creates a seamless bitstream in which codewords can be decoded without regard for input word boundaries.

By isolating these functions into separate planes, the decoder ensures that input alignment and bit merging occur concurrently with symbol decoding. For example, while the OR plane is supplying data to the priority encoder for prefix detection, the input plane can already be rotating and preparing the next input word for decoding. This pipelined structure prevents idle cycles and reduces the dependency between operations, thereby improving decoding efficiency and overall throughput. Moreover, the separation enables predictable behaviour under varying symbol lengths and Rice parameters. Whether the prefix spans 5 bits or 25 bits, and whether the suffix begins in the same word or the next, the decoder can continue processing without stalling. This capability is critical in embedded systems, where predictable latency and resource determinism are often mandatory.

The use of plane separation also simplifies error handling and synchronization. If a codeword is partially available in the current window, the OR plane signals the input plane to supply additional bits. This coordination maintains stream integrity and ensures that symbol decoding is never attempted on incomplete data. The architecture thus not only supports high-speed decoding but also maintains robustness across varying input conditions.

4.2.4. Decoding Flow

The Rice decoding process in this architecture is implemented as a tightly controlled pipeline that integrates the priority encoder and plane separation into a high-performance workflow. Each cycle begins by loading two 32-bit word into the OR plane and one 32-bit word into the input plane appended by 32 zeroes. This word is shifted using a barrel shifter so that the first undecoded bit aligns with the decoding window. Meanwhile, the OR plane holds leftover bits from the previous cycle and merges them with the new input word to form a full decoding buffer. This buffer provides a continuous and correctly aligned segment of the compressed stream. The decoder then uses the priority encoder to scan this buffer for the first occurrence of a ‘1’. The number of preceding ‘0’s is recorded as the quotient q. With the prefix length known, the decoder shifts the buffer by q + 1 bits to skip the unary portion and delimiter. The next k bits are then extracted as the binary suffix r (if J = 1) or again fed

to the priority encoder (if J > 1). These bits are obtained using a mask and shift operation since k is fixed per block and readily available from the block header.

After retrieving both q and r, the decoder reconstructs the mapped value using the by the operation ***(q<<k)|r*** to form the final symbol. This operation, performed using a left shift and an addition, yields the original prediction error. The mapped value is passed to the inverse mapping logic, which converts it into a signed error according to the rules defined in the CCSDS 121.0-B-3 specification.

Next, the signed error is added to the predicted value (typically the previous sample) to recover the original uncompressed data sample. Simultaneously, the OR plane shifts out all the bits consumed during this cycle, and the decoder updates its internal pointer. If the remaining buffer in the OR plane is insufficient to decode the next symbol, the input plane is signalled to load and align the next input word. This operation is coordinated without stalls or external control intervention.

The decoding flow supports a continuous stream of Rice-coded data. It tolerates misaligned input, dynamically adjusts for symbol length, and handles cross-boundary decoding transparently. The decoder operates block by block, with each block's Rice parameter and format defined by a header. Once a block is complete, the decoder resets its internal registers and waits for the next block's configuration, maintaining full compliance with the CCSDS 121.0-B-3 standard.

The Carry Generator block (Fig. 4.1) is responsible for monitoring the available bit length in the OR plane and signalling when it falls below the required threshold for decoding. When this condition is met, the Control Path block takes over and generates the necessary control signals to manage the decoding process. Specifically, it issues load signals for the OR plane and Input plane and enables the Priority Encoder based on the values of J, K, and the number of symbols decoded. The coordination between these blocks ensures that decoding proceeds seamlessly, even when codewords span across multiple input boundaries.

This pipeline ensures that decoding is deterministic and efficient, even under varying data patterns or symbol distributions. Every component—from prefix detection to buffer shifting—is executed in a single or pipelined clock cycle, resulting in a decoder capable of high-throughput performance on platforms such as FPGAs or ASICs.

5. PACKET HEADER PARSER

In CCSDS-based telemetry systems, each data stream is typically composed of a fixed-size header followed by compressed payload data. The header contains essential metadata such as synchronization markers, packet identification, and control fields necessary for downstream decompression and processing. In this design, the packet header is 48 bits wide and must be isolated from the incoming stream before further decoding. The remaining portion of the stream—termed *CDS data*—contains the actual

compressed content to be processed. To perform this separation and expose the compressed data cleanly to the decompression pipeline, a **Packet Header Parser** module is implemented. The function of this module is to detect, buffer, and remove the first 48 bits of the incoming data stream and then forward the remaining data, one 32-bit word at a time, as valid packet data.

The parser operates on 32-bit input data chunks, received alongside a *datavalid* signal that indicates the presence of valid data on a given clock cycle. Internally, the module maintains a 48-bit buffer, which accumulates the first two 32bit input words and stores the header. Once the header is captured, the parser forwards the remaining portion of the buffer (i.e., bits [47:16]) as *packetdata* and asserts a *pvalid* flag to signal downstream modules that valid data is available for further processing.

6. CDS HEADER PARSER

In the CCSDS 121.0-B-3 standard, once the initial 48-bit packet header has been removed, the remaining bitstream consists of a sequence of compressed data segments. Each of these segments begins with a compact header that encodes essential control information, including the block size, Rice parameter, and mode selection. These headers are embedded within the first few bits of each compressed block and must be extracted and interpreted before the actual decompression process can begin.

The **CDS Header Parser** module is responsible for identifying and extracting these embedded control fields. It also aligns the bitstream to ensure that only the compressed payload is passed on to the decoder. The header length is determined dynamically based on the value of the parameter n. The data is buffered into a FIFO, allowing the decoder to access it flexibly and on demand. This buffering mechanism decouples data reception from decompression, enabling efficient handling of variablelength codes and supporting continuous, high-throughput operation of the decoding pipeline.

7. INVERSE MAPPING AND PREDICTION

The Inverse Mapping and Prediction block is responsible for reconstructing the original uncompressed data samples from the Rice-decoded values produced earlier in the decompression pipeline. It performs two key functions in accordance with the CCSDS 121.0-B-3 standard. First, it executes the inverse mapping step, converting the unsigned Rice-decoded value back into a signed prediction error. This step handles both standard and exceptional cases, accounting for the dynamic threshold parameter , which is derived from the previously reconstructed sample. The mapping ensures proper recovery of both positive and negative errors based on a conditional transformation logic defined in the standard.

$$\Delta = \begin{cases} \delta / 2 & if\ \delta \leq 2\theta\ and\ \delta\ is\ even \\ -(\delta + 1) / 2 & if\ \delta \leq 2\theta\ and\ \delta\ is\ odd \\ \delta - \theta & if\ \delta > 2\theta\ and\ \theta = \hat{x} - \hat{x}\,min \\ -(\delta - \theta) & if\ \delta > 2\theta\ and\ \theta = \hat{x}\,max\hat{x} - \hat{x} \end{cases}$$

Second, the block applies a 1D prediction algorithm, which reconstructs the original sample by adding the recovered error to a reference—typically the most recently reconstructed sample. Modular arithmetic is used to ensure that the result remains.

8. DESIGN ARCHITECTURE

8.1. Decompressor Architecture

The figure (Fig 8.1) illustrates the overall architecture of the CCSDS 121.0-B-3 compliant decompressor, highlighting the key dataflow stages from packet reception to final sample reconstruction. The system begins with the **Packet Header Parser**, which extracts and removes the 48-bit primary packet header from the incoming CCSDS stream. This header contains high-level metadata and framing information required to identify valid data blocks. The remaining payload is forwarded to the **CDS Header Parser**, which interprets the secondary header fields to determine blockspecific parameters such as the compression mode, k value, block length, and alignment offsets.

Following header parsing, the actual compressed data—now properly framed and indexed—is fed into a **FIFO buffer**, which acts as a staging area to provide steady and aligned access to the decompression engine. This buffering ensures that the decoder has continuous access to compressed bits without stalling, despite potential variability in input timing or burstiness in data flow.

The **Decoder** block (explained in section **9.2**) forms the core of the decompression pipeline. It implements bitstream parsing and decoding logic tailored to the selected compression mode, specifically supporting **fixed-length (FS) and splitsample Rice-coded** segments as described in the CCSDS standard. The decoder separates the compressed input into prediction error symbols (either full-length or quotient-remainder pairs) and forwards these to the next stage.

The final stage is the **Reverse Mapping and Prediction** block, which reconstructs the original data samples from the decoded prediction errors using 1D predictor logic. This logic uses a previously decoded or reference value (xref) and computes each new sample using inverse Rice mapping. The resulting stream, labeled

Original Data, is a fully reconstructed version of the input telemetry or scientific data, now decompressed and ready for further mission processing or storage.

Together, these blocks form a modular and pipeline-friendly decompression system that adheres to CCSDS 121.0-B-3 specifications. Each stage is responsible for a distinct functional role, allowing for independent testing, verification, and potential reuse in future telemetry or payload data systems.

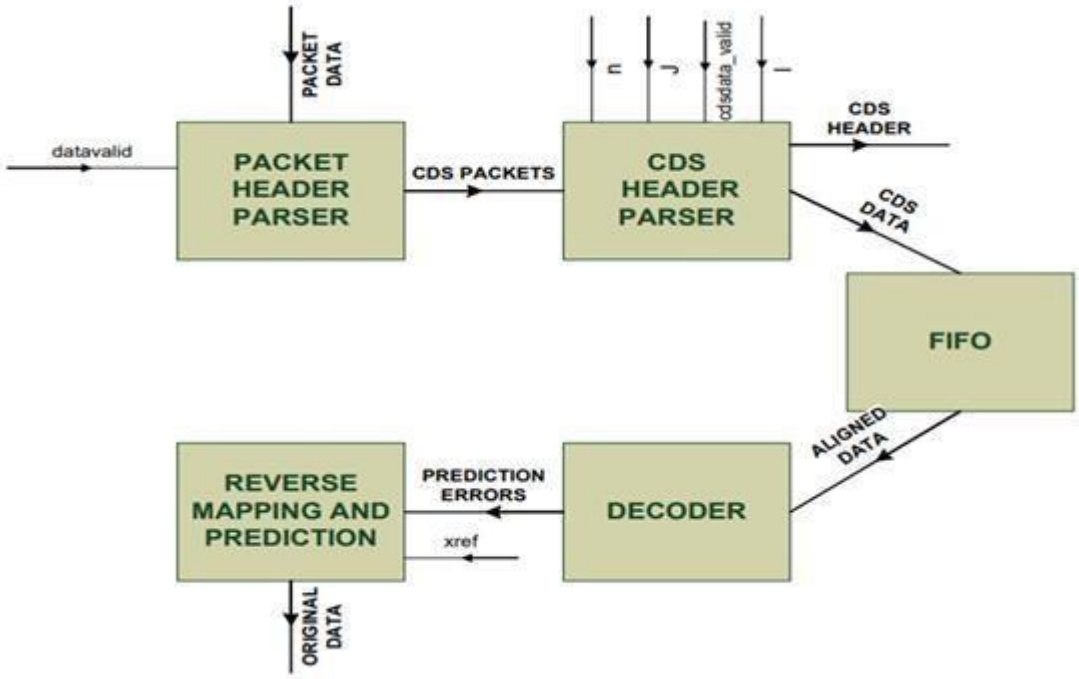

Fig 8.1: Decompressor Architecture

8.2. Decoder Architecture:

The architecture consists of six core modules:

- **OR Plane**
- **Input Plane**
- **Carry Generator**
- **Control Path**
- **Symbol Generator**

Each of these modules is responsible for a specific stage in the decompression flow. The **OR Plane** performs bit-aligned decoding of unary and Rice-coded segments from the incoming bitstream. The **Input Plane** ensures that the OR plane receives a continuously aligned stream of data, rotating and aligning input blocks as necessary based on current decoding phase and symbol length. The **Carry Generator** monitors the remaining bits available in the OR plane and signals whether additional data needs to be loaded before the next symbol can be decoded.

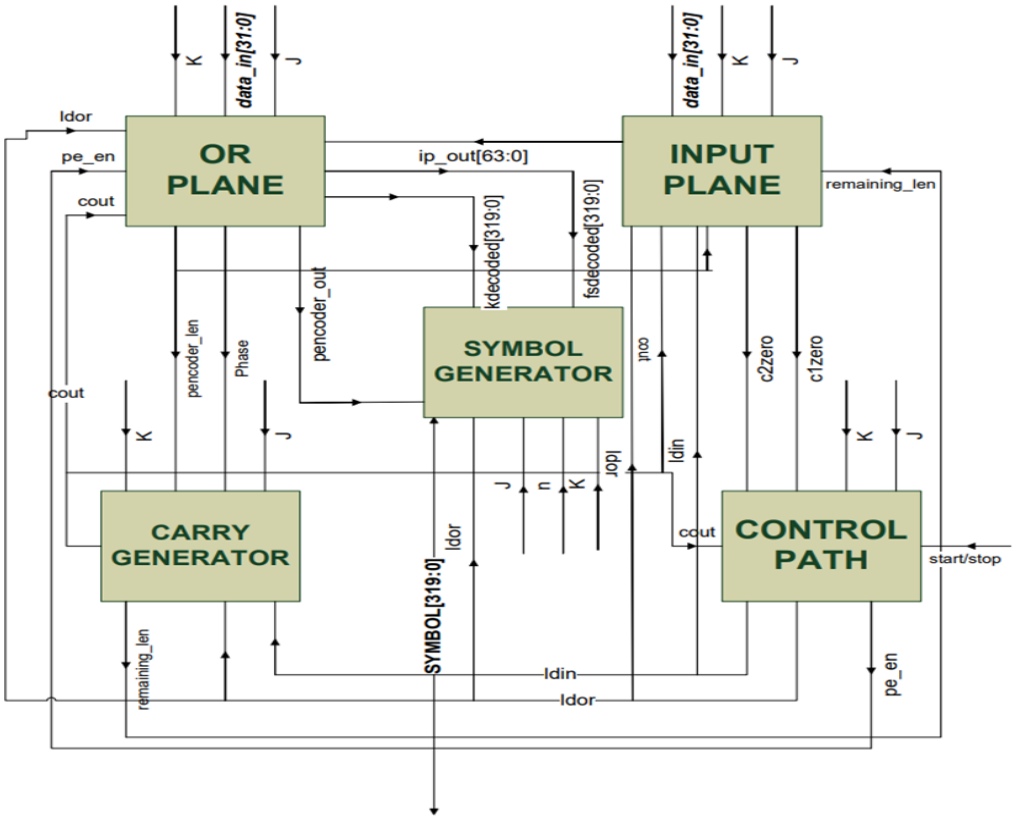

Fig 8.2: Block diagram of Rice Decoder **8.2.1.**

Input Plane:

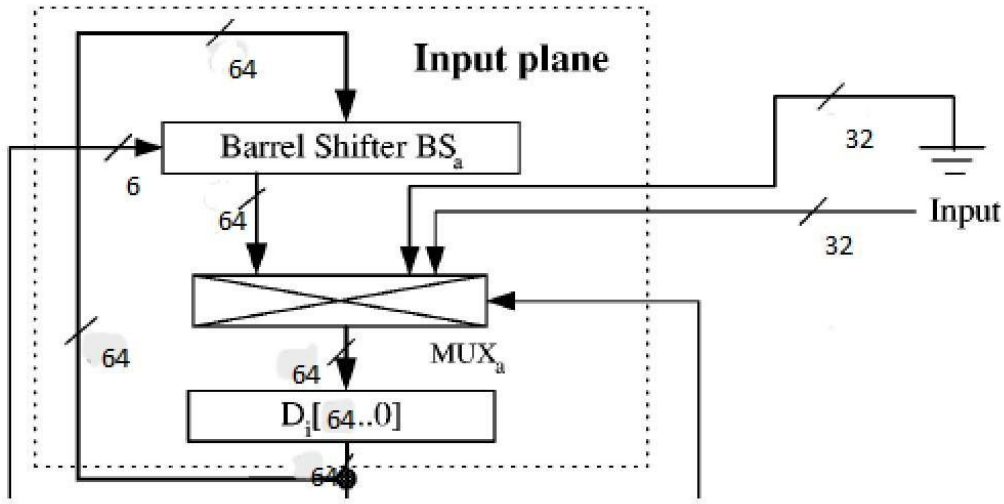

Fig 8.2.1: Input Plane Architecture

The **Input Plane** serves as the interface between the incoming compressed data and the decoding logic. Its primary function is to prepare and align the input data stream before it is passed to the OR plane for decoding. To ensure decoding accuracy and bitlevel alignment, the Input Plane includes mechanisms for both data loading and controlled bit rotation.

As illustrated in the architecture diagram (Fig. 9.1.1), the module receives a 32-bit input word which is extended to 64 bits. This 64-bit extended input is processed through a **Barrel Shifter** labeled BS_a, which performs a left rotation based on a programmable shift amount. This shift aligns the input data relative to previously consumed bits, ensuring proper bitstream continuity across block boundaries.

The shifted output from the barrel shifter is then sent to a **multiplexer** (MUX_a), which selects between the newly aligned input and a rotated version of the previously stored 64-bit word based on *carry* signal generated. A 64-bit register $D_i[64..0]$ holds the final selected output of the multiplexer, which is then made available to the OR plane for decoding. This register is updated every clock cycle based on phase logic, ensuring that the data flow keeps pace with decoding operations.

The architecture is designed to support two decoding phases — fixed-sample and remainder (Rice) — and uses dynamic shift amounts depending on the current phase. This flexibility allows the input plane to maintain bitstream alignment and continuity regardless of symbol lengths or coding mode transitions.

8.2.2. OR Plane:

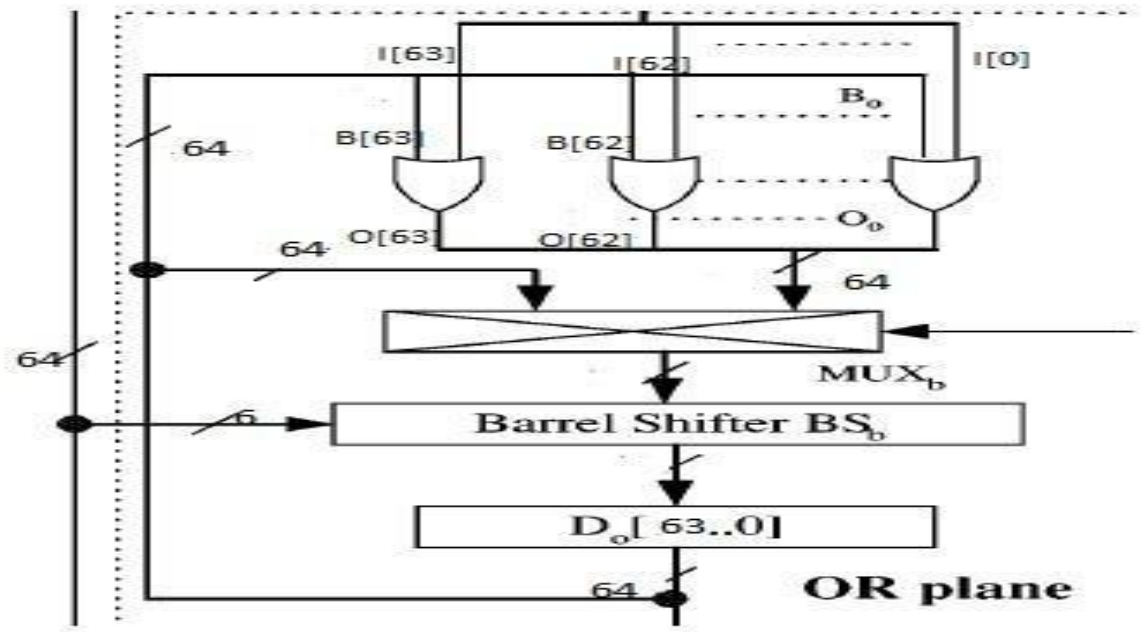

Fig 8.2.2: OR Plane Architecture

The **OR Plane** is responsible for decoding the compressed bitstream into its constituent fixed-sample (FS) and Rice-coded remainder values. Its architecture is designed to manage dynamic bit extraction, controlled shifting, and symbol separation across decoding phases. OR plane includes a 64-bit data register Do[63:0] that holds the active bitstream window. This register is fed by a **barrel shifter**, which allows for left-rotation of the current data word based on the length of the symbol being decoded (either the prefix length in the FS phase or the Rice parameter k in the remainder phase).

Before the data enters the barrel shifter, it passes through a **multiplexer** which selects between two input sources — either the original 64-bit data from the input plane or a logic-combined stream derived from **bitwise AND** and **OR** operations performed on the current input III and an auxiliary bit mask Bo. This allows for controlled substitution or merging of incoming bits depending on the decoding context.

The result of the barrel-shifted output is stored in Do, which continuously updates and feeds the decoder logic with a correctly aligned data window. This rotating and updating mechanism ensures that the correct bits are made available for decoding symbols in sequence without corruption or misalignment.

The OR plane operates under the control of the **carry signal** (cout) and decoding **phase** (FS or K), both of which determine the barrel shifter's rotation amount and the type of symbol currently being processed. The output of this stage is used directly by the **priority encoder** in the FS phase and passed to bitmasking logic in the remainder phase for Rice decoding.

8.2.3. Carry Generator:

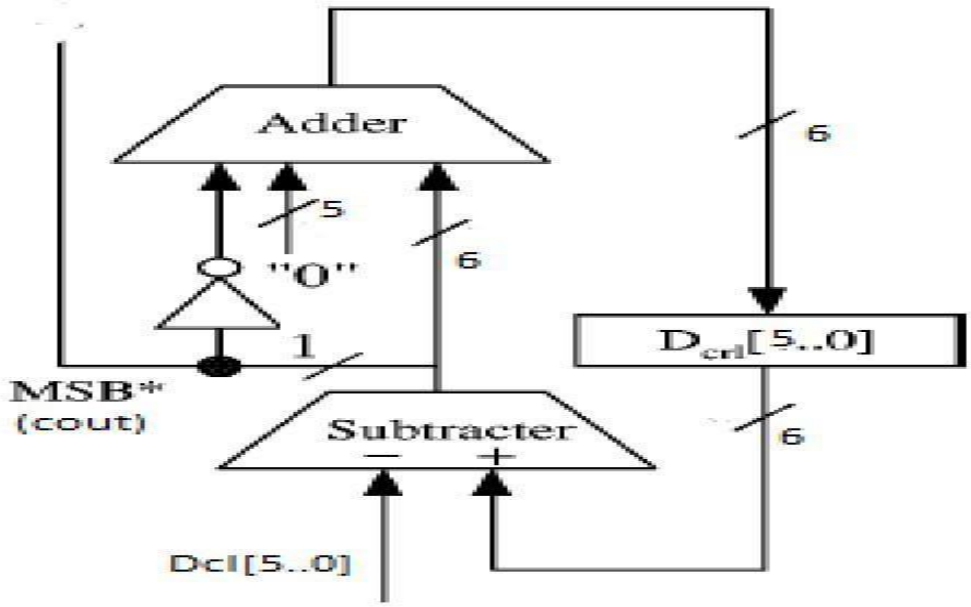

Fig 8.2.3: Carry generation

The carry generator consists of three main computational blocks: a **subtractor**, an **adder**, and two **registers** Dcrl[5:0] and Dcl[5:0]. It operates on two 6-bit values — the current **code length** (either the Rice parameter k or FS prefix length) and the number of **remaining bits** in the OR plane.

The **subtractor** computes the difference between the remaining bits and the code length. The most significant bit (MSB) of the result determines whether the subtraction yielded a negative value, indicating insufficient bits to decode the next symbol. This MSB is directly interpreted as the control signal cout (carry out), which is used by other modules — including the input plane and control path — to initiate loading of new data.

The output of the subtractor is also passed to an **adder** along with a logic value (either "0" or "1") derived from the inversion of the MSB. This combination allows for conditional correction of the result using two's complement logic, ensuring accurate calculation of updated remaining length. The final output is stored in the register Dcrl, which holds the updated **remaining length** for use in the next cycle.

8.2.4. Control Path:

The **control path** plays a crucial role in orchestrating the overall operation of the decompression process. It functions as the centralized decision-making unit that

governs the sequence of actions across the decoder pipeline, including data loading, decoding phase transitions, and completion.

As illustrated in the state diagram (Fig. X), the control path operates as a finite state machine (FSM), managing the flow of control through various states such as initialization, input/output loading, fixed-sample and remainder decoding, conditional branching, and finalization. The system begins in an initialization state, ensuring that all registers and control signals are reset and prepared for new input. It then transitions through structured stages of data alignment and decoding, depending on the current phase and status flags.

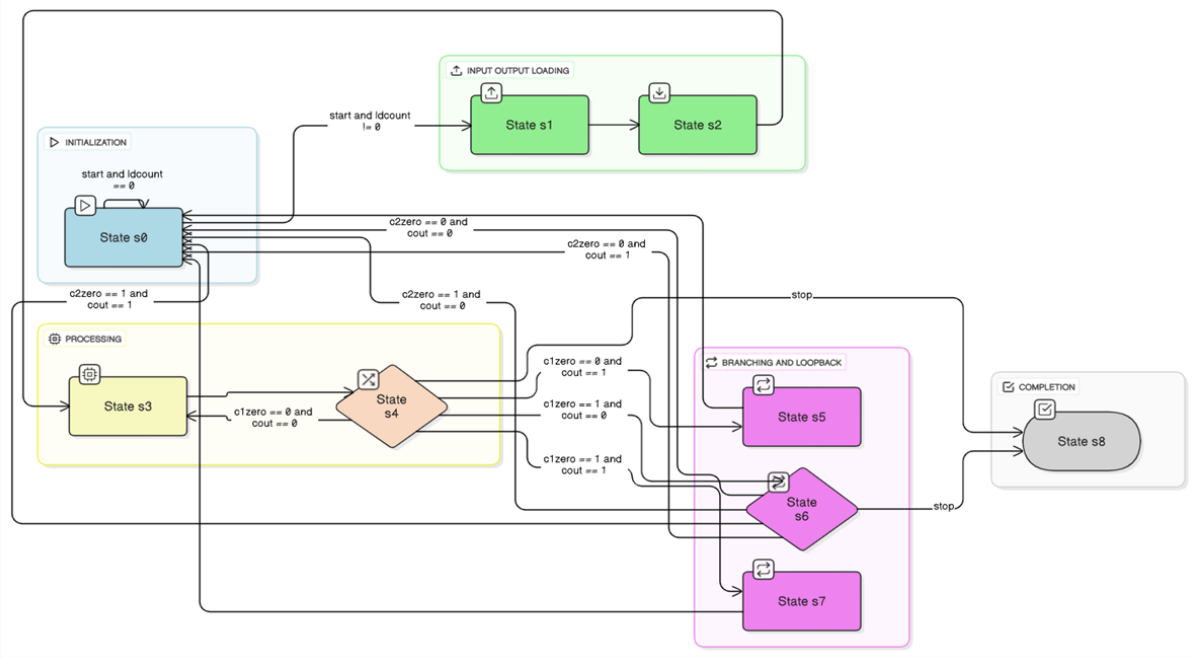

Fig 8.2.4: Control-path of Rice Decoder

9. SIMULATION RESULTS

9.1. OR PLANE DECODING AND INPUT PLANE ROTATION:

The waveform shown in Fig. 5.1 illustrates the interaction between the **OR Plane (do)** and the **Input Plane (ipp)** during the decoding process. The do signal represents the OR Plane output, which holds the aligned bitstream window currently under evaluation by the priority encoder. As the decoder operates, the priority encoder scans the do data to determine the unary prefix length and compute the quotient for the Rice code. This operation is triggered every cycle based on the availability of valid data and decoding enable conditions.

The ipp signal corresponds to the Input Plane buffer, which shifts or rotates based on the current decoded codeword length (tracked internally as cle). Each time a codeword is successfully decoded, the Input Plane rotates the next 32-bit word into position, aligning the stream for the subsequent decoding operation. This interaction ensures seamless handling of variable-length codes, even when symbols span across word boundaries.

From the waveform, it can be observed that as the value of cle changes (representing the number of bits consumed), the Input Plane shifts accordingly to align the next decoding window. Simultaneously, the OR Plane updates the data (do) visible to the priority encoder, ensuring that the decoding window always starts at the correct position. The correct functioning of the priority encoder and Input Plane coordination is verified by the progressive decoding of values shown in the output sequence (do), where each decoded quotient is correctly extracted based on the incoming stream.This illustrates the correctness of the decoding logic and the effectiveness of the plane separation technique, which enables concurrent data alignment and decoding operations without stalling or misalignment.

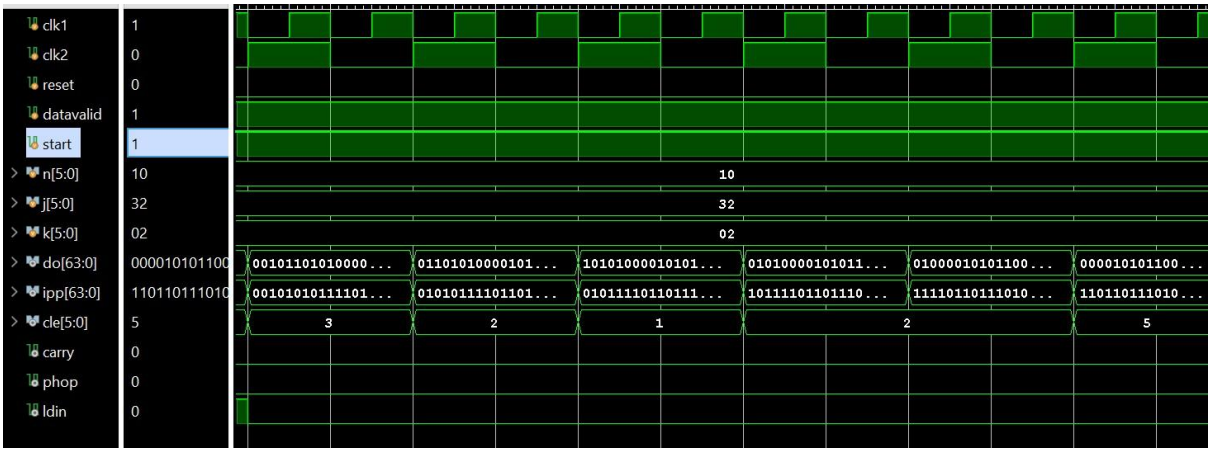

Fig 9.1: Decoding operation and dataflow

9.2. CARRY GENARATION AND INPUT PLANE LOADING:

The waveform demonstrates the behavior of the carry generation logic, which plays a critical role in managing the interaction between the OR Plane (do) and the Input Plane (ipp). This logic is responsible for tracking the number of remaining valid bits available in the OR Plane and generating a carry-out signal (carry) when this bit count falls below a predefined threshold—typically the length of a 32-bit word.

The signal rem_length represents the remaining number of bits in the OR Plane that are still undecoded. As decoding progresses, and codelength (the bit length of the current decoded symbol) is consumed, rem_length is decremented accordingly. Once rem_length drops below the threshold (e.g., 32 bits), the carry

signal is asserted, prompting the system to initiate a load of new data into the Input Plane.

This operation is reflected in the waveform where, at specific points (e.g., when rem_length decreases to 31, 29, etc.), the carry signal goes high, and subsequently, the ld_in signal is asserted. The assertion of ld_in triggers the Input Plane to fetch and align the next 32-bit word, which is then merged with the remaining bits in the OR Plane to maintain a continuous bitstream for decoding.

The updated data in ipp and its corresponding effect on do ensure that the decoding window is seamlessly extended across word boundaries. This logic guarantees that the decoder never attempts to read past available bits, thus maintaining decoding accuracy and preventing data corruption.

In summary, this carry generation mechanism acts as a threshold-based refill controller for the OR Plane. It ensures that decoding is not interrupted due to insufficient bitstream availability and maintains smooth, continuous processing of variable-length codes.

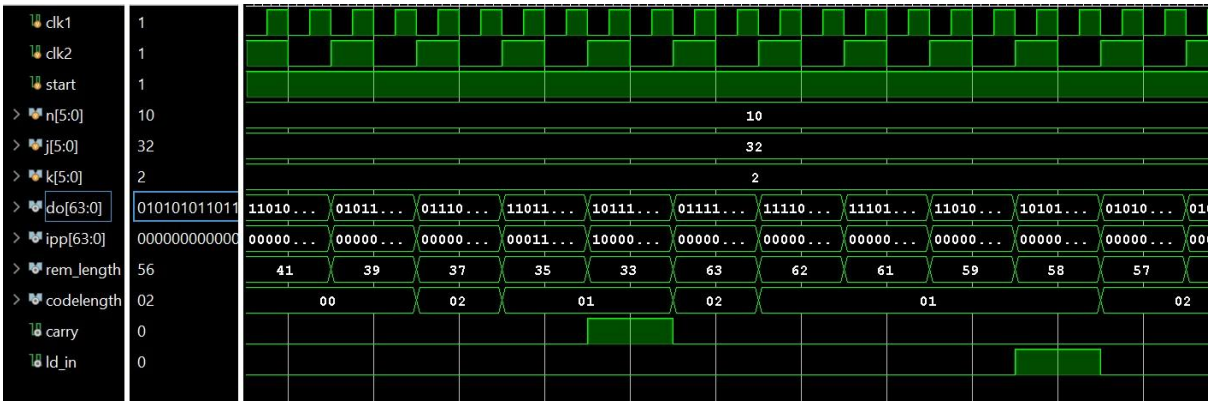

Fig 9.2: Carry generation and OR plane merging

9.3. CONTROL PATH:

The waveform in Fig. 5.3 illustrates the role of the **Control Path** in coordinating the decoding process by issuing control signals based on the decoding state and available resources. The control path is responsible for managing signal sequencing across major components—specifically, enabling the **priority encoder (pe_en)**, triggering new word loading into the **Input Plane (ld_in)**, and monitoring the status of **fixed-length (count_fs)** and **Rice decoding (count_k)** symbol counters.

In this simulation, the signals count_k and count_fs track the number of decoded symbols in a Rice-coded or fixed-length coded block, respectively. Once either counter reaches the total number of samples (n), decoding for the current block is considered complete. The signal fs_zero and k_zero indicate when the

respective count has reached zero, which the control path uses to disable the decoder or transition to the next phase of operation.The **pe_en (priority encoder enable)** signal is asserted only when decoding is active and not yet complete (i.e., fs_counter has not reached zero). This ensures that the priority encoder is only active during valid decoding windows, conserving power and avoiding unnecessary logic transitions.

The **carry** signal, as explained earlier, is asserted when the number of remaining bits in the OR Plane falls below a specified threshold (typically 32). When this occurs, the control path responds by asserting the **ld_in** signal, which triggers the Input Plane to fetch and align the next 32-bit word. This coordination ensures that the OR Plane always maintains a sufficient number of bits for continuous decoding.In the waveform, it can be observed that ld_in is asserted in response to carry going high, and pe_en is toggled according to the decoding state driven by count_k. This logic enables seamless integration between decoding, bitstream management, and symbol tracking.

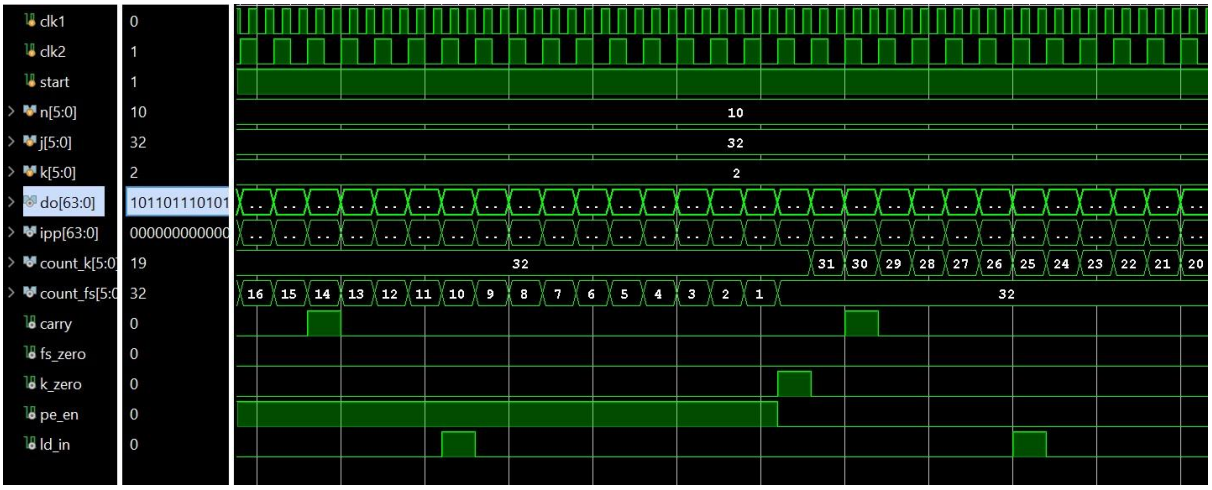

Fig 9.3: Signal generation by Control Path

9.4. SYMBOL FORMATION:

The waveform shown in Fig. A illustrates the final stage of the decompression pipeline, where the outputs of the fixed-length and Rice decoders are combined to form the complete decompressed data symbols. The signals fsdecoded and kdecoded represent the intermediate outputs from the fixed-length decoder and the Rice decoder, respectively. These outputs contain partial sample data depending on the selected compression mode for each block as defined by the CCSDS 121.0-B-3 standard.

A Symbol Generator module is responsible for merging these outputs into a unified data word, referred to as symbol. The merging is handled according to the active compression mode—either fixed-sample or split-sample coding. In fixed-length mode, the decoded values are loaded directly from fsdecoded,

whereas in split-sample (Rice-coded) mode, the symbol is assembled using both MSBs from the fixed-length path and LSBs recovered through Rice decoding via kdecoded.

From the waveform, it is observed that both fsdecoded and kdecoded are populated with valid values in their respective phases. Once a complete symbol is formed, it is output through the symbol signal, which serves as the reconstructed data word corresponding to the original uncompressed sample. The symbols maintain 1:1 alignment with the input blocks and reflect correct reconstruction behavior, confirming the functional correctness of the decompression logic. This final step ensures that all processing—ranging from bitstream alignment, unary decoding, inverse mapping, and prediction—is consolidated into a coherent and usable output stream. The symbol output can then be forwarded to storage, visualization, or further processing stages depending on the system's application.

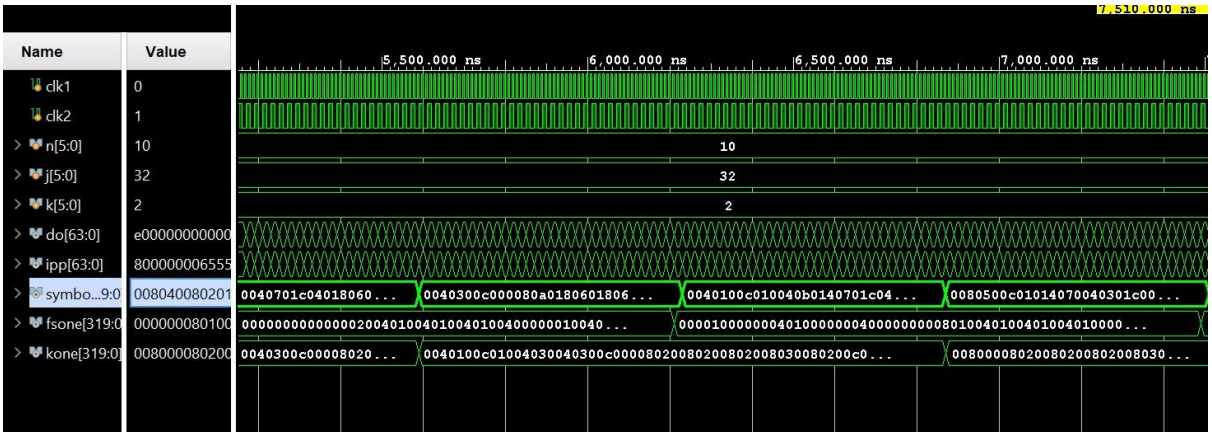

Fig 9.4: Symbol formation from fs_codes and k_codes

9.5. Packet Header Parser:

The waveform in Fig. B demonstrates the operation of the Packet Header Parser module, which is responsible for extracting the initial 48-bit header from the incoming CCSDS-compliant packet stream. This module plays a critical role in isolating the metadata required for downstream decompression, such as mode selection, block configuration, and sync verification, before passing the remaining data on to the decoding pipeline.

The signal data represents the 32-bit input word arriving from the stream interface. During the initial phase, datavalid remains low, and the module captures two consecutive 32-bit inputs into a 48-bit buffer register. This is done over two clock cycles, during which the buffer accumulates the first 48 bits of the incoming data. As shown in the waveform, once the buffer has fully captured

the header, the signal packethead is updated with the complete 48-bit header value.

Following this, datavalid is asserted, indicating that valid payload data is now present at the input. At this point, the module shifts its operation from header buffering to direct data forwarding. Incoming 32-bit data words are routed directly to the packetdata output, bypassing the buffer, as observed in the aligned values of packetdata and data after the header is captured.

This mechanism ensures that the decompression pipeline receives a clean separation between the control metadata (the 48-bit header) and the compressed payload data. It also aligns with the CCSDS standard's requirement for fixed-size headers followed by arbitrarily sized data fields. The module supports pipelined operation and is designed to transition from header accumulation to data delivery without requiring external intervention or reset.

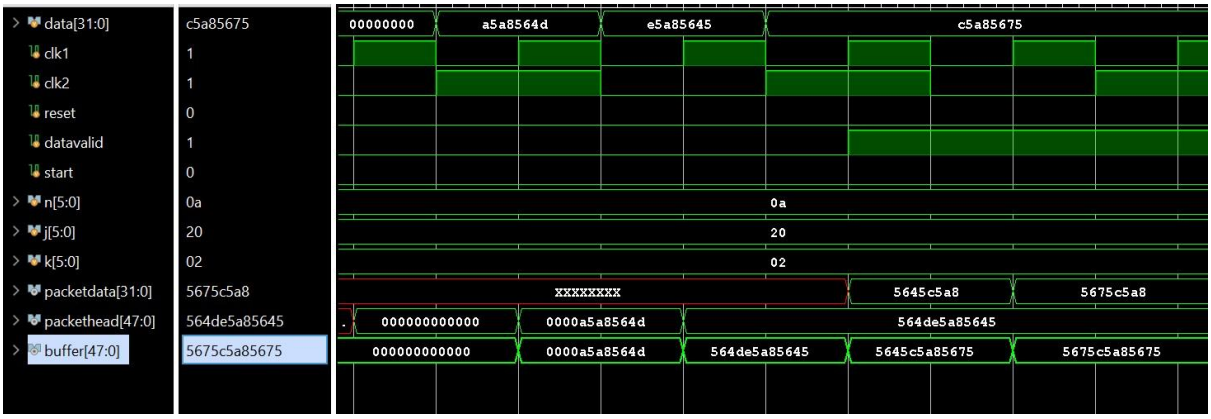

Fig 9.5: Packet Header Parser

9.6. Inverse Prediction and Mapping:

The waveform in Fig. C demonstrates the operation of the **Inverse Mapping module**, which performs the final stage of decompression by reconstructing the original sample values from the decoded Rice-coded symbols. This operation is essential to recover the true data values from the residual error representation used during compression.

In accordance with CCSDS 121.0-B-3 and the Rice coding logic, each Rice-decoded symbol represents a residual error (δ) with respect to a previously predicted sample (xref or x_prev). The inverse mapping function converts this residual back to the original data sample by applying a deterministic, reversible transformation. This transformation uses a 1D prediction scheme and a conditional reconstruction rule based on a computed threshold value, referred

to as θ (theta). Theta is derived from the predicted value, which determines the symmetry of error mapping.

In the simulation, the symbol input consists of 32 individual 10-bit delta values packed into a 320-bit input stream. The module iteratively applies the inverse mapping rule:

- If the delta is within 2θ, it is mapped to a signed error using the Rice standard's even/odd rule.
- If the delta exceeds 2θ, it is interpreted as a linear distance from the threshold. The reconstructed sample is computed as:

The output signal xout shows the packed original data samples reconstructed by the module. In the waveform, the output starts populating a few cycles after reset, as the module sequentially processes each symbol over multiple clock cycles. The consistent and stable appearance of the output word confirms the correct mapping from compressed deltas to original values.

Fig 9.6: Retrieving original data from symbols

10. FUTURE SCOPE

The current implementation presents a functional and efficient decompression system targeting a subset of the CCSDS 121.0-B-3 lossless compression standard, specifically the fixed-length and split-sample modes utilizing Rice coding with 1D prediction. While the architecture meets essential compliance and performance goals, there remains considerable scope for future development and extension—particularly in terms of hardware realization, scalability, and robustness for real-world satellite systems.

The current design employs a 1D prediction model, which is lightweight and computationally efficient. However, for higher-dimensional sensor data, such as multispectral images or scientific payload outputs, future work can

explore the integration of more complex prediction techniques—such as 2D predictive models, gradient-adaptive prediction, or median-edge detectors. These methods are particularly suited for spatially correlated data and can significantly enhance compression efficiency when paired with a corresponding encoding strategy.

From a hardware standpoint, the presented decompressor is well-suited for **FPGA or ASIC-based implementation**, making it a viable candidate for **onboard satellite data handling units**. Future work could focus on synthesizing the design for specific FPGA families (e.g., Xilinx Ultrascale+, Microsemi RTG4, Intel Cyclone V) or space-grade ASICs, taking into account constraints such as power consumption, logic area, and radiation hardness. Optimization techniques such as **resource sharing**, **clock gating**, **loop unrolling**, and **pipeline balancing** can be explored to enhance throughput and reduce critical path delay, allowing for real-time decoding of high-throughput telemetry streams.

Additionally, deeper hardware integration could involve **direct interfacing with satellite communication subsystems**, enabling the decompressor to act as an intermediate block between the telemetry receiver and data storage modules. Support for AXI or AMBA bus protocols, DMA-based memory access, and clock-domain crossing logic would improve compatibility with system-on-chip (SoC) platforms. Furthermore, integrating on-chip buffering and stream arbitration logic can help the decompressor handle bursty or packetized inputs, such as those found in CCSDS Transfer Frame streams or multiplexed sensor payloads.

Another critical avenue is the **inclusion of fault-tolerant design mechanisms**, which are essential for space-grade electronics. As satellite systems are often exposed to ionizing radiation and other harsh environmental factors, the decompressor can be hardened by design using techniques such as **triple modular redundancy (TMR)**, **ECCprotected memory**, and **synchronous reset logic**. Error detection methods like **CRC validation**, **header verification**, and **bitstream alignment monitoring** can also be incorporated to ensure data integrity and system reliability under single-event upset (SEU) conditions.

To support verification and validation, a **co-simulation framework** involving both RTLlevel decompressor modules and high-level CCSDS-compliant software models can be developed. This would allow for thorough functional coverage, corner-case testing, and bit-accurate validation of the hardware implementation. Additionally, integrating the decompressor into a **closed-loop compression–decompression pipeline**, alongside a custom or reference CCSDS encoder, would facilitate end-to-end testing, benchmarking, and round-trip fidelity analysis.

Finally, the design can be extended with a **software interface or GUI toolchain** for system integrators. This interface could allow engineers to monitor internal registers, view decoded outputs in real-time, visualize the Rice decoding phase, and trace control signals. Such features would significantly ease debug, prototyping, and mission-specific tuning of the decompressor when embedded into larger satellite platforms.

In summary, while the presented architecture forms a reliable and efficient baseline for CCSDS decompression, there are multiple meaningful directions for future work. These include algorithmic extensions (advanced prediction, mode support), architectural enhancements (pipelining, resource efficiency), and full hardware deployment (synthesis, validation, fault tolerance). With these extensions, the design could evolve into a robust, space-ready IP core for next-generation satellite data systems.

11. REFERENCES

1. CCSDS 121.0-B-3: "*Lossless Data Compression- Blue Book*"

2. Jae Ho Jeon, Young Seo Park, and Hyun Wook Park, "*A Fast Variable-Length Decoder Using Plane Separation.*"

3. UNIVERSIDAD DE LAS PALMAS DE GRAN CANARIA "*CCSDS Lossless Data Compression IP-Core Space Applications.*"

4. Shang Xue and Bengt Oelmann, "*Parallel Variable-Length Decoder Architecture for Alternated Coded GR-Codes*"

5. CCSDS 133.0-B-2: "*Space Packet Protocol"*

6. NATIONAL AND KAPODESTRIAN UNIVERSITY OF ATHENS, "*HARDWARE IMPLEMENTATIONS OF CCSDS 121.0-B-2 LOSSLESS DATA COMPRESSION STANDARD IN FPGA*"

7. CCSDS 120.0-G-4: “*Lossless Data Compression- Green Book*”

8. Shang Xue and Bengt Oelmann “EFFICIENT VLSI IMPLEMENTATION OF A VLC DECODER FOR GOLOMB-RICE CODE USING ALTERNATING CODING”, Department of Information Technology and Media, Mid Sweden University