VISVESVARAYA TECHNOLOGICAL UNIVERSITY, BELGAUM

A Project Report on

"LOSSLESS HYPERSPECTRAL DATA COMPRESSION USING CCSDS 123.0-B-1 FOR SATELLITE IMAGING"

Thesis submitted in the partial fulfillment of the requirements for the award of the degree of

Bachelor of Engineering

In

ELECTRONICS AND COMMUNICATION ENGINEERING

For the year 2016-2017

K. M. AISHWARYA	**(1RN13EC063)**
RACHANA R	**(1RN13EC115)**
PREETI M. SOBARAD	**(1RN13EC110)**

Under the guidance of

Dr. VIPULA SINGH
Head of Department

Mrs. SOWMYA P
Scientist 'SE'
ISAC, ISRO, Bengaluru

2016 – 17

DEPARTMENT OF ELECTRONICS AND COMMUNICATION ENGINEERING
R.N.S. INSTITUTE OF TECHNOLOGY
BENGALURU – 560 098

R.N.S. INSTITUTE OF TECHNOLOGY

Channasandra, Bengaluru - 560 098

DEPARTMENT OF ELECTRONICS AND COMMUNICATION ENGINEERING

CERTIFICATE

This is to certify that the thesis work entitled "**LOSSLESS HYPERSPECTRAL DATA COMPRESSION USING CCSDS 123.0-B-1 FOR SATELLITE IMAGING**" is a bonafide work carried out by **K. M. Aishwarya (1RN13EC063), Rachana R (1RN13EC115) and Preeti M. Sobarad (1RN13EC110)** students of **R.N.S. Institute of Technology, Bengaluru** in partial fulfillment for the award of the degree of **Bachelor of Engineering** in **Electronics and Communication Engineering** of **Visvesvaraya Technological University, Belgaum** during the academic year of 2016-17.

It is certified that all suggestions indicated for Internal Assessment have been incorporated in the report deposited in the Department Library. The project report has been approved as it satisfies the academic requirements in respect of project work prescribed for the said Degree.

Dr. Vipula Singh
(Head of Department& Internal Guide)

Dr. M. K. Venkatesha
(Principal)

External Viva

Name of the Examiners:	**Signature with Date:**
1. …….	……
2. …….	……

भारत सरकार
अन्तरिक्ष विभाग
इसरो उपग्रह केन्द्र
हवाई पत्तन मार्ग, विमानपुर डाक घर
बेंगलूर–560 017

GOVERNMENT OF INDIA
DEPARTMENT OF SPACE
ISRO SATELLITE CENTRE
HAL AIRPORT ROAD, VIMANAPURA POST
BANGALORE - 560 017

DATE : 31/05/2017

Certificate

This is certify that **K M Aishwarya**

of RNS Institute of Technology, Bangalore *has undergone*

Project training in Data Handling & Storage Group *at this Centre*

during the period from 19/01/2017 *To* 28/04/2017

COURSE : Bachelor of Engineering (Electrconics & communication Engineering)

PROJECT TITLE : Lossless Hyperspectral Data Compression using CCSDS 123.0-B-1 for Satellite Imaging

PROJECT MEMBERS : Rachana R, Preeti M Sobarad

PERFORMANCE : Excellent

ISRO SATELLITE CENTRE HRDD PPEG BANGALORE

(Prakash K S)
Head HRD Division, ISAC

(Rajendra Hulyal)
Group Head PPEG, ISAC

भारत सरकार
अन्तरिक्ष विभाग
इसरो उपग्रह केन्द्र
हवाई पत्तन मार्ग, विमानपुर डाक घर
बेंगलूर–560 017

GOVERNMENT OF INDIA
DEPARTMENT OF SPACE
ISRO SATELLITE CENTRE
HAL AIRPORT ROAD, VIMANAPURA POST
BANGALORE - 560 017

DATE : 31/05/2017

Certificate

This is certify that **Rachana R**

of RNS Institute of Technology, Bangalore *has undergone*

Project training in Data Handling & Storage Group *at this Centre*

during the period from 19/01/2017 *To* 28/04/2017

COURSE : Bachelor of Engineering (Electrconics & communication Engineering)

PROJECT TITLE : Lossless Hyperspectral Data Compression using CCSDS 123.0-B-1 for Satellite Imaging

PROJECT MEMBERS : K M Aishwarya, Preeti M Sobarad

PERFORMANCE : Excellent

ISRO SATELLITE CENTRE HRDD PPEG BANGALORE

(Prakash K S)
Head HRD Division, ISAC

(Rajendra Hulyal)
Group Head PPEG, ISAC

भारत सरकार
अन्तरिक्ष विभाग
इसरो उपग्रह केन्द्र
हवाई पत्तन मार्ग, विमानपुर डाक घर
बेंगलूर–560 017

GOVERNMENT OF INDIA
DEPARTMENT OF SPACE
ISRO SATELLITE CENTRE
HAL AIRPORT ROAD, VIMANAPURA POST
BANGALORE - 560 017

DATE : 31/05/2017

Certificate

This is certify that **Preeti M Sobarad**

of RNS Institute of Technology, Bangalore *has undergone*

Project training in Data Handling & Storage Group *at this Centre*

during the period from 19/01/2017 *To* 28/04/2017

COURSE : Bachelor of Engineering (Electrconics & communication Engineering)

PROJECT TITLE : Lossless Hyperspectral Data Compression using CCSDS 123.0-B-1 for Satellite Imaging

PROJECT MEMBERS : K M Aishwarya, Rachana R

PERFORMANCE : Excellent

ISRO SATELLITE CENTRE HRDD PPEG BANGALORE

(Prakash K S)
Head HRD Division, ISAC

(Rajendra Hulyal)
Group Head PPEG, ISAC

R.N.S. INSTITUTE OF TECHNOLOGY

Channasandra, Bengaluru - 560 098

DEPARTMENT OF ELECTRONICS AND COMMUNICATION ENGINEERING

DECLARATION

We, **K. M. Aishwarya, Rachana R, Preeti M. Sobarad,** students of B.E. **Electronics and Communications Engineering**, R. N. S. Institute of Technology, Bengaluru, hereby declare that the entire work embodied in this project report titled **"Lossless Hyperspectral Data Compression using CCSDS 123.0-B-1 for Satellite Imaging"** submitted to **Visvesvaraya Technological University**, Belagavi, has been carried out by us at ISAC, ISRO, Bengaluru under the guidance of **Mrs. Sowmya P** and **Dr. Vipula Singh**. This report has not been submitted in part or full for the award of any diploma or degree of this or any other University.

K. M. AISHWRYA
1RN13EC063

RACHANA R
1RN13EC115

PREETI. M. SOBARAD
1RN13EC110

ACKNOWLEDGEMENT

The joy and satisfaction that accompany the successful completion of any task would be incomplete without thanking those who made it possible.

We consider ourselves proud to be a part of RNS Institute of Technology, the institution which molded us in all our endeavors.

We express our deep gratitude to **Dr. R N Shetty**, for providing state of art facilities.

We express our deep gratitude to our beloved Chairman **Dr. H N Shivashankar**, Director, who has always been a great source of inspiration. We would like our express our sincere thanks to **Dr. M K Venkatesha**, Principal, for their valuable guidance and encouragement throughout our program.

We express our profound gratitude to the coordinators who have given valuable suggestions and guidance throughput the project. We would like to express our sincere gratitude to our internal guide **Dr. Vipula Singh**, Professor and Head of Department, for her guidance, continuous support and motivation in completing the project successfully.

Our special thanks to **Mr. Sura PS**, Head of Division, Data Handling and Storage Group (DHSG) at ISAC-ISRO, for providing us with an opportunity to carry out the project at ISRO.

We convey our profound gratitude towards our external guide **Mrs. Sowmya P**, Scientist - S'E' at ISAC-ISRO for being a constant source of support, encouragement and guidance throughout the course of the project.

We are also thankful to all the teaching and non-teaching faculty of RNSIT and Data Handling Systems Lab of the DHSG Department at ISRO for their constant support.

ABSTRACT

The payload data acquired by remote sensing space missions are voluminous and require larger transmission bandwidth, memory and longer data transmission time to ground station. Hence onboard compression is crucial for optimal usage of onboard storage and downlink bandwidth.Remote sensing hyperspectral data consist of hundreds of contiguous spectral bands which inturn demands larger downlink bandwidth and onboard storage as opposed to single band payload data. The higher interband correlation among the spectral bands in hyperspectral images is utilized for maximal compression performance.

Consultative Committee for Space Data Systems (CCSDS) has recommended a novel lossless hyperspectral image compression standard, 123.0-B-1 intended to be suitable for onboard spacecraft with lesser computational complexity and memory requirements.

This project work aims at software implementation of CCSDS 123.0-B-1 algorithmin VHDL, analyzing different algorithm parameters with respect to compression parameters and validation of the data. The software implementation results are verified and compared from the results produced in Microsoft Visual Studio 2015. For the software implementation of the algorithm, the entire compression algorithm has been fragmented into multiple modules, with the predictor and encoder being two main entities of the compression standard. The hyperspectral image is fed as an input to the predictor whose output serves as a driving input to the encoder. The encoder output is a compressed hyperspectral image. Both the predictor and encoder codes comprise of several software modules in cascade. Finally the entire code has been integrated with the help of a Top Module in VHDL, which receives its input signal drivers from the Top Module Testbench. The VHDL coding has been done in Libero Version 9.2 which is a product of ActelMicrosemi®. Finally the compression parameters are evaluated for different performance evaluation parameters and the variation in the resulting Compression Rate (CR) is recorded.

ORGANIZATION PROFILE

The **Indian Space Research Organization** is the primary space agency of India. ISRO is amongst the largest government space agencies in the world. Its primary objective is to advance space technology and use its applications for national benefit. Established in 1969, ISRO superseded the erstwhile Indian National Committee for Space Research (INCOSPAR). Headquartered in Bangalore, ISRO is under the administrative control of the Department of Space, Government of India.

ISRO has achieved numerous milestones ever since it's established. It built India's first satellite, Aryabhata, which was launched by the Soviet Union on 19 April in 1975. In 1980, Rohini became the first satellite to be placed in orbit by an Indian-made launch vehicle, SLV-3. ISRO subsequently developed two other rockets: the Polar Satellite Launch Vehicle (PSLV) for launching satellites into polar orbits and the Geosynchronous Satellite Launch Vehicle (GSLV) for placing satellites into geostationary orbits. These rockets have launched numerous communications satellites and earth observation satellite. On 22 October in 2008, Chandrayaan-1, India sent its first mission to the Moon. Over the years, ISRO has conducted a variety of operations for both Indian and foreign clients. ISRO's future plans include indigenous development of GSLV, manned space missions, further lunar exploration, mars exploration and interplanetary probes. ISRO has several field installations as assets, and cooperates with the international community as a part of several bilateral and multilateral agreements. 5 November 2013, ISRO launched its Mars Orbiter Mission (MOM), which is currently en route to Mars. All launches are carried out from ISRO SHAR center at Sriharikota. ISRO has the following centres.

1. Vikram Sarabhai Space Centre (VSSC), Thiruvananthapuram, is engaged in design and development of satellite launch vehicles.
2. ISRO Satellite Centre (ISAC), Bangalore, is engaged in design and development of satellites for multiple users as communication, weather forecasting and remote sensing etc.

3. SatishDhawan Space Centre (SDSC/SHAR), Sriharikota, is the operational base of ISRO. It is fully equipped with sophisticated launch pad vehicles.
4. ISRO Telemetry Tracking and Commanding Network (ISTRAC), Bangalore, provides ground support for the launch vehicle and satellite mission of ISRO.
5. Space Application Centre (SAC), Ahmadabad, is engaged with activities in telecommunications and development of payloads for spacecraft.
6. Liquid Propulsion Systems Centre (LPSC) , Bangalore , is engaged with the activities in launch vehicle propulsion system engine developing and testing.
7. National Remote Sensing Agency (NRSC), Hyderabad, is engaged in developing and utilizing modern remote sensing techniques.
8. Master Control Facility (MCF), Hassan, is engaged in monitoring and control of INSAT series of spacecrafts from different ground stations.

CONTENTS

LIST OF FIGURES

LIST OF TABLES

CHAPTER 1
INTRODUCTION

CHAPTER 1

INTRODUCTION

1.1 HYPERSPECTRAL IMAGES

The most significant recent breakthrough in remote sensing has been the development of hyperspectral sensors and software to analyze the resulting image data. Fifteen years ago only spectral remote sensing experts had access to hyperspectral images or software tools to take advantage of such images. Over the past decade hyperspectral image analysis has matured into one of the most powerful and fastest growing technologies in the field of remote sensing.

The “hyper” in hyperspectral means “over” as in “too many” and refers to the large number of measured wavelength bands. The wavelengths may be separated by filters or by the use of instruments that are sensitive to particular wavelengths, including light from frequencies beyond the visible light range, such as infrared. Spectral imaging can allow extraction of additional information which the human eye fails to capture with its receptors for red, green and blue. It was originally developed for space-based imaging. The wavelengths may be separated by filters or by the use of instruments that are sensitive to particular wavelengths, including light from frequencies beyond the visible light range, such as infrared. Spectral imaging can allow extraction of additional information which the human eye fails to capture with its receptors for red, green and blue. It was originally developed for space-based imaging. A hyperspectral image is depicted in Fig. 1.1.Hyperspectral data are a challenge for data compression. Several factors make the constraints particularly stringent and the challenge exciting. First is the size of the data: as a third dimension is added, the amount of data increases dramatically making the compression necessary at different steps of the processing chain. Also different properties are required at different stages of the processing chain with variable tradeoff. Second, the differences in spatial and spectral relation between values make the more traditional 3D compression algorithms obsolete. And finally, the high expectations from the scientists using hyperspectral data require the assurance that the

compression will not degrade the data quality. All these aspects are investigated in the present chapter and the different possible tradeoffs are explored.

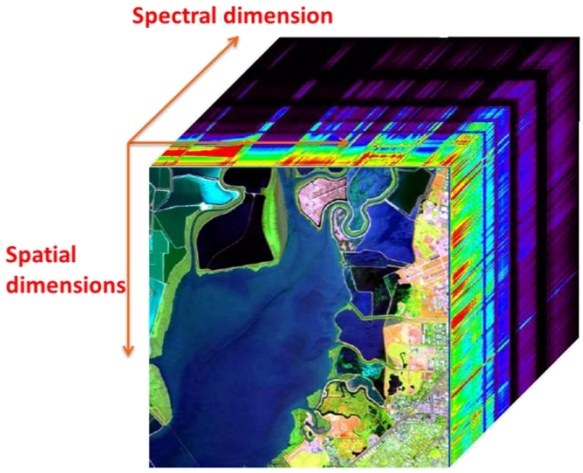


Fig. 1.1 Hyperspectral Image

Hyperspectral images are spectrally over-determined, which means that they provide ample spectral information to identify and distinguish spectrally unique materials. Hyperspectral imagery provides the potential for more accurate and detailed information extraction than possible with any other type of remotely sensed data. [1]

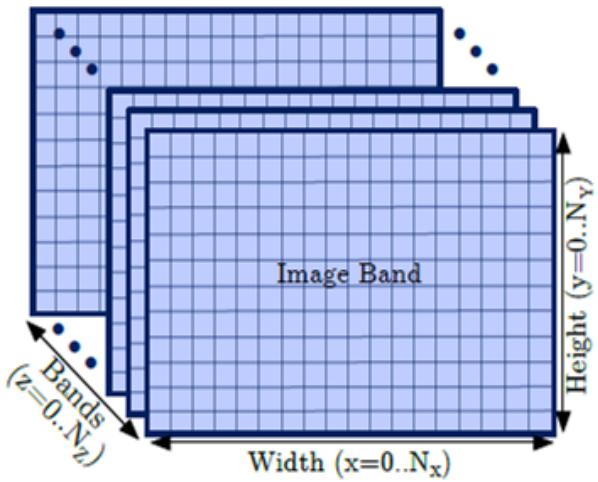


Fig. 1.2 HSI Structure

The structure of hyperspectral image is illustrated in Fig 1.2 .Each image represents a narrow wavelength range of the electromagnetic spectrum, known as a spectral band. These images

are combined to form a three-dimensional(x, y, z) hyperspectral data cube (see Fig.1.1) for processing and analysis, where *x* and *y* represent two spatial dimensions of the scene, and *z* represents the spectral dimension (comprising a range of wavelengths).***A collection of spectral images over several wavelength intervals for the same scene is called a multispectral or hyper spectral image.***

The details not visible to the human eye can be determined from these additional wavelengths and would be useful while predicting the weather and studying geology. Hyperspectral images contain two kinds of correlations namely intra-band correlation and inter-band correlation. Intra-band correlation is the correlation between nearby pixels in the same band and inter-band correlation is the correlation between pixels in adjacent bands.

1.2 SENSORS FOR HYPERSPECTRAL DATA COLLECTION

Hyper spectral sensors collect information as a set of 'images'. Each image represents a range of the electromagnetic spectrum and is also known as a spectral band. These 'images' are then combined and form a three-dimensional hyper spectral data cube for processing and analysis. Hyper spectral cubes are generated from airborne sensors like Airborne Visible/Infrared Imaging Spectrometer (AVIRIS). The precision of these sensors is typically measured in spectral resolution, which is the width of each band of the spectrum that is captured. If the scanner detects a large number of fairly narrow frequency bands, it is possible to identify objects even if they are only captured in a handful of pixels.

Multispectral remote sensors such as the Lands at Thematic Mapper and SPOT XS produce images with a few relatively broad wavelength bands. Hyperspectral remote sensors, on the other hand, collect image data simultaneously in dozens or hundreds of narrow, adjacent spectral bands. These measurements make it possible to derive a continuous spectrum for each image cell, as shown in the illustration below. After adjustments for sensor, atmospheric, and terrain effects are applied, these image spectra can be compared with field or laboratory reflectance spectra in order to recognize and map surface materials such as particular types of vegetation or diagnostic minerals associated with ore deposits.

Hyperspectral images contain a wealth of data, but interpreting them requires an understanding of exactly what properties of ground materials we are trying to measure, and how they relate to the measurements actually made by the hyperspectral sensor.

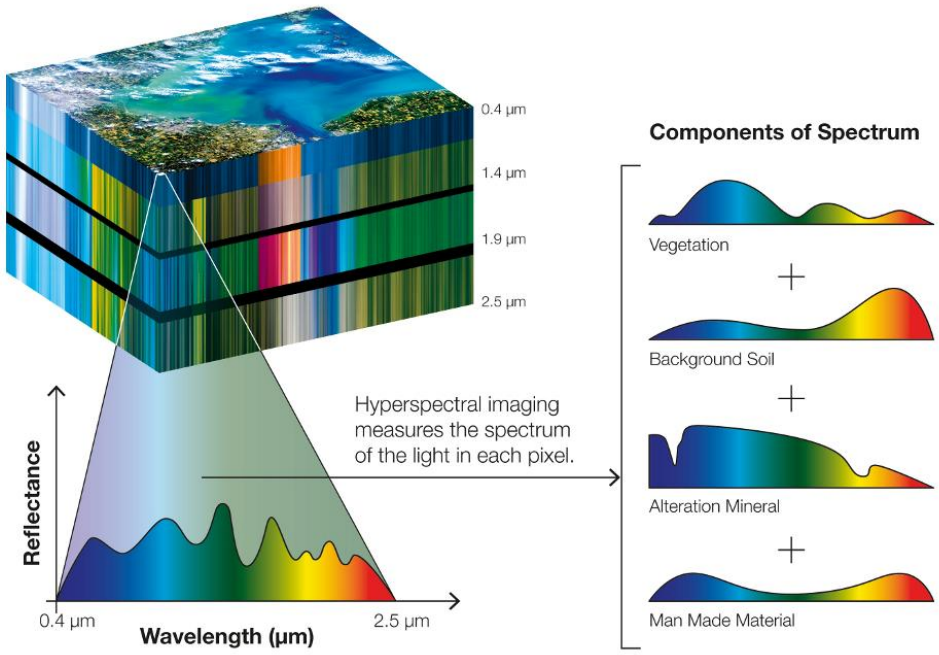


Fig 1.3 Hyperspectral Imaging Technology

Remote sensing imagers are designed to measure the reflectance of areas on the Earth's surface. Reflectance is defined as the percentage of the light hitting a material that is then reflected by that material, and can be represented across a range of wavelengths, showing a pattern or spectrum, which can uniquely identify certain materials. Each pixel in a hyperspectral image is composed of hundreds of reflectance values which define their spectral signature as shown in Fig 1.3.

The hyperspectral scanners are of two main types namely whiskbroom and pushbroom as depicted in Fig 1.4. Whiskbroom scanners or across track scanners reflect light into a single detector using a mirror which moves back and forth to collect measurements from one pixel in the image at a time. On the other hand, pushbroom scanners, or along track scanners, use a line of detectors arranged perpendicular to the light direction of the spacecraft. The image is collected one line at the time as the spacecraft flies forward. The selection of the type of

scanner depends mainly on the purpose and specific requirements of the remote sensing mission.

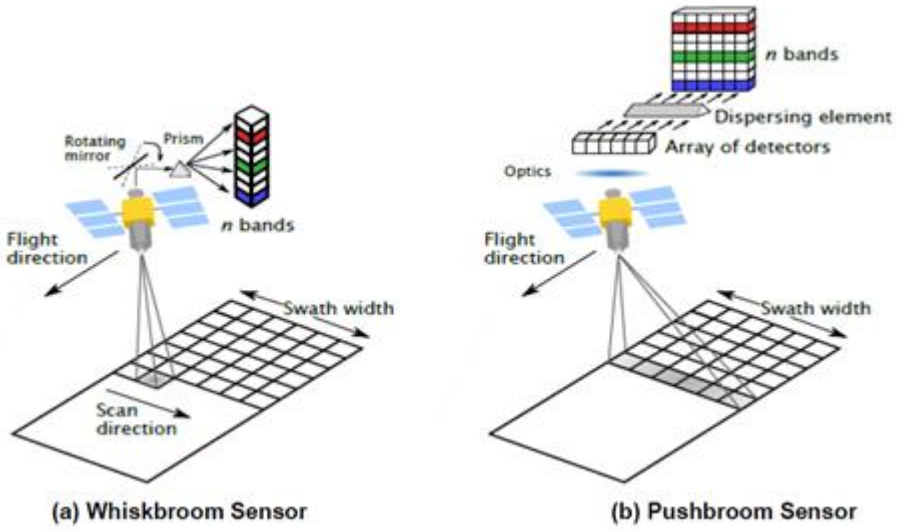


Fig 1.4 Hyperspectral Scanners Used Onboard

However, spatial resolution is a factor in addition to spectral resolution. However, the huge volume of data for hyper spectral images causes difficulties in transmission and storage. Undoubtedly, data compression is a necessary and effectual solving approach.

1.3 DATA COMPRESSION – ITS SIGNIFICANCE

In signal processing, data compression, source coding, or bit-rate reduction involves encoding information using fewer bits than the original representation. Compression can be either lossy or lossless. Lossless compression reduces bits by identifying and eliminating statistical redundancy. No information is lost in lossless compression. Lossy compression reduces bits by removing unnecessary or less important information. The process of reducing the size of a data file is referred to as data compression. In the context of data transmission, it is called source coding (encoding done at the source of the data before it is stored or transmitted) in opposition to channel coding.

Compression is useful because it reduces resources required to store and transmit data. Computational resources are consumed in the compression process and, usually, in the

reversal of the process (decompression). Data compression is subject to a space–time complexity trade-off. For instance, a compression scheme for video may require expensive hardware for the video to be decompressed fast enough to be viewed as it is being decompressed, and the option to decompress the video in full before watching it may be inconvenient or require additional storage. The design of data compression schemes involves trade-offs among various factors, including the degree of compression, the amount of distortion introduced (when using lossy data compression), and the computational resources required to compress and decompress the data.

Data compression is needed because it allows the data to be stored in an area without taking up an unnecessary amount of space. Data compression uses a series of algorithms to reduce the amount of real space that the data would normally take up.

The amount of data that is shown when it is compressed is dependent on how the data has been compressed. If there is an extremely large amount of data in a sequence, it will be compressed to a large size. If there is a small amount of data that is compressed, it will not take up as much real space as compressed large amounts of data. Determining the amount of space that the data will take up is dependent on the algorithm that is used to compress the data. If the algorithm is used properly and formatted specifically to the data, the data can take up nearly no real space. If a generalized algorithm is used, the data may not fit into a compressed area as well as if a specific algorithm is used. The algorithm is the determining factor on how well the data will compress, how much space it will take up and how much of the data will be available after compression.

1.4 COMPRESSION TYPES: LOSSY, LOSSLESS AND NEAR-LOSSLESS

Lossless compression algorithms enable the users to retrieve exactly the original data from the compressed bit stream. They are generally based on a predictor followed by an entropy coder of the residuals. The most recent publications in this domain converge towards a compression ratio around 3:1. Such a compression ratio is insufficient to meet the constraint for onboard systems.

However, they are highly relevant for archiving the data and distribution to the end-user. Lossy compression on the contrary introduces a distortion in the data and the original data cannot be retrieved in its exact form. These methods generally have parameters that can be adjusted to move along the rate–distortion curve. Reducing the bit rate increase the distortion and vice-versa. When the distortion remains small, the algorithm can be qualified as near-lossless. Two main definitions appear in the literature for near-lossless compression of hyperspectral data. The first definition considers that the compression is near-lossless if the noise it introduces remains below the sensor noise: the data quality remains the same. The other definition considers that an algorithm is near-lossless if the distortion is bounded. We will stick to the former definition which guaranties no distortion from the application point of view: the compression remains in the noise of the sensor.

By definition, **lossless** image compression is a compression algorithm that allows the original image to be *perfectly reconstructed* from the original data. **Lossy** image compression is a type of compression where a certain amount of information is *discarded* which means that some data are lost and hence the image cannot be decompressed with 100% originality. **Near Lossless** compression is when the noise introduced remains *below the sensor noise*: the data quality remains the same.

1.5 LOSSLESS COMPRESSION OF HYPERSPECTRAL IMAGES

Concerned with the effects of global climate change NASA initiated the Mission to Planet Earth enterprise in 1991. Many of the efforts of NASA and other national space agencies are coming to fruition. Among these efforts has been the development of remote sensing instruments with high levels of spatial and spectral resolutions. The products generated by these sensors promise to revolutionize our understanding of climatology, meteorology, and land management. As the amount of data generated by these sensors is enormous and as the number of sensors continues to grow it is clear that the role of data compression will be crucial in this development. At different points in the path from the sensor to the end-user the compression needs will be different. It can be either Lossy, Lossless or Near-Lossless.

These techniques have been successful in part because the human visual system is insensitive to certain kinds and levels of distortion. Thus selective loss of information can be used to significantly enhance the compression performance. However, hyperspectral images are not solely intended to be viewed by human beings. They are usually processed for various applications such as feature extraction, classification, target detection, and object identification. Any distortion introduced during the compression process can get amplified by the processing and have a significant deleterious effect on the application. As such, lossy compression is not always acceptable. At a minimum lossless compression is needed if the data is to be stored for some period of time on the acquisition platform and for transmission from the acquisition platform to the ground station. Lossless compression is also needed for data archiving.

Compression schemes, and in particular lossless compression schemes rely on statistical structure in the data. In hyperspectral images there are, loosely speaking, two kinds of correlations that can be exploited. One is spatial correlation between adjacent pixels in a spectral band, and the other is correlation between pixels in adjacent bands. The spatial correlation can be easily exploited using compression method developed for standard grayscale or colour images. However, how to efficiently make use of the redundancy between adjacent bands for high resolution hyperspectral images is still an open question, and will largely determine how much further compression gain can be obtained. Most compression techniques proposed for lossless hyperspectral image compression fall into two categories: vector quantization and predictive coding. Lossless data compression has been suggested for many space science exploration mission applications either to increase the science return or to reduce the requirement for on-board memory, station contact time, and data archival volume. A Lossless compression technique guarantees full reconstruction of the original data without incurring any distortion in the process. The Lossless Data Compression technique recommendation preserves the source data accuracy by removing redundancy from the application source data. In the decompression processes the original source data is reconstructed from the compressed data by restoring the removed redundancy. The reconstructed data is an exact replica of the original source data. The quantity of redundancy

removed from the source data is variable and is highly dependent on the source data statistics, which are often non-stationary.

1.6 APPLICATIONS OF HYPERSPECTRAL IMAGES

Hyperspectral remote sensing is used in a wide array of applications. Hyperspectral images contain ton of information, surface information and its spectrum behaviour should be understood deeply. This type of images are finding their importance in different fields as before it was just used for remote sensing application. Here are few applications of hyperspectral images.

1) AGRICULTURE -Although the cost of acquiring hyperspectral images is typically high, for specific crops and in specific climates, hyperspectral remote sensing use is increasing for monitoring the development and health of crops. In Australia, work is under way to use **imaging spectrometers** to detect grape variety and develop an early warning system for disease outbreaks. Furthermore, work is underway to use hyperspectral data to detect the chemical composition of plants, which can be used to detect the nutrient and water status of wheat in irrigated systems. Fig 1.5 shows the degree of infection in wheat plant.

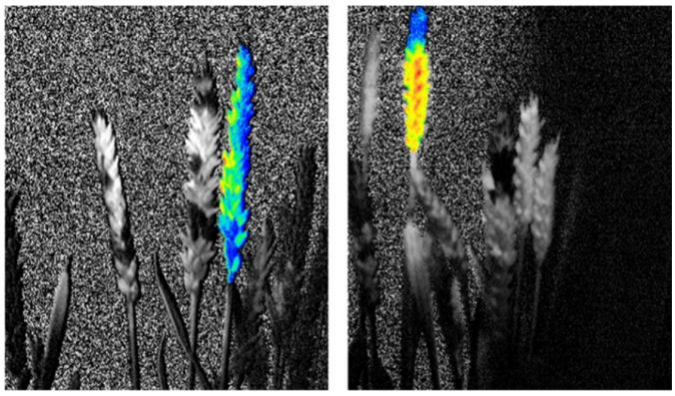


Fig. 1.5 Early detection of plant diseases - Wheat heads infected at largely different degrees.

Seed Viability Study: By using the hyperspectral image and plotting the reflectance spectrum one can conclude that whether those seed are viable or not viable. Seed might be

looking same through naked eyes but its viability will be trace down by the hyperspectral image.

Another application in agriculture is the **detection of animal proteins** in compound feeds to avoid bovine spongiform encephalopathy (BSE), also known as mad-cow disease. Different studies have been done to propose alternative tools to the reference method of detection, (classical microscopy). One of the first alternatives is near infrared microscopy (NIR), which combines the advantages of microscopy and NIR. In 2004, the first study relating this problem with hyperspectral imaging was published. Hyperspectral libraries that are representative of the diversity of ingredients usually present in the preparation of compound feeds were constructed. These libraries can be used together with chemometric tools to investigate the limit of detection, specificity and reproducibility of the NIR hyperspectral imaging method for the detection and quantification of animal ingredients in feed.

Fig 1.6 shows the Hyperspectral camera embedded on OnxyStar HYDRA-12 UAV from AltiGator.

Fig. 1.6 Onyxstar HYDRA-12 UAV with embedded hyperspectral camera for agricultural research

2) MEDICAL DIAGNOSE -Early disease detection and disease prevention are very important for the healthy body. Hyperspectral imaging technology can be used to detect the early of various types of **cancer or retinal disease**.

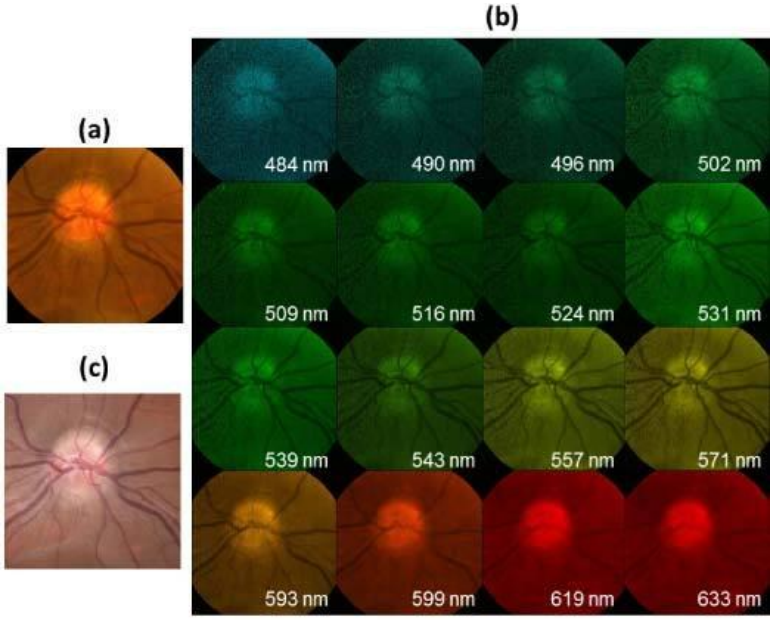


Fig. 1.7 Snapshot hyperspectral retinal camera with the IMS

Fig 1.7 shows the Snapshot of hyperspectral retinal camera with the Image Mapping Spectrometer (IMS).

Eye care: Researchers at the Université de Montréal are working with Photon etc. and Optina Diagnostics to test the use of hyperspectral photography in the diagnosis of retinopathy and macular edema before damage to the eye occurs. The **metabolic hyperspectral camera** will detect a drop in oxygen consumption in the retina, which indicates potential disease. An ophthalmologist will then be able to treat the retina with injections to prevent any potential damage.

3) FOOD - Hyperspectral imaging is widely used in the food sector. It is used in different discipline of food industry, bruise detection in apples, freshness of the fish, citrus fruit inspection, distribution of sugar in melons, and sorting of potatoes. For example **apple bruise** is not visible on the early stage and it takes few days to show dark colour mark. In

this type of scenario hyperspectral imaging techniques can be used to track the early stage of bruise for the quality control.

In the **food processing industry**, hyperspectral imaging, combined with intelligent software, enables digital sorters (also called optical sorters) to identify and remove defects and foreign material (FM) that are invisible to traditional camera and laser sorters. By improving the accuracy of defect and FM removal, the food processor's objective is to enhance product quality and increase yields. The sorter's software compares the hyperspectral images collected to user-defined accept/reject thresholds, and the ejection system automatically removes defects and foreign material.

The recent commercial adoption of hyperspectral sensor-based food sorters is most advanced in the nut industry where installed systems maximize the removal of stones, shells and other foreign material (FM) and extraneous vegetable matter (EVM) from walnuts, pecans, almonds, pistachios, peanuts and other nuts. This is represented in the Fig 1.8. Here, improved product quality, low false reject rates and the ability to handle high incoming defect loads often justify the cost of the technology.

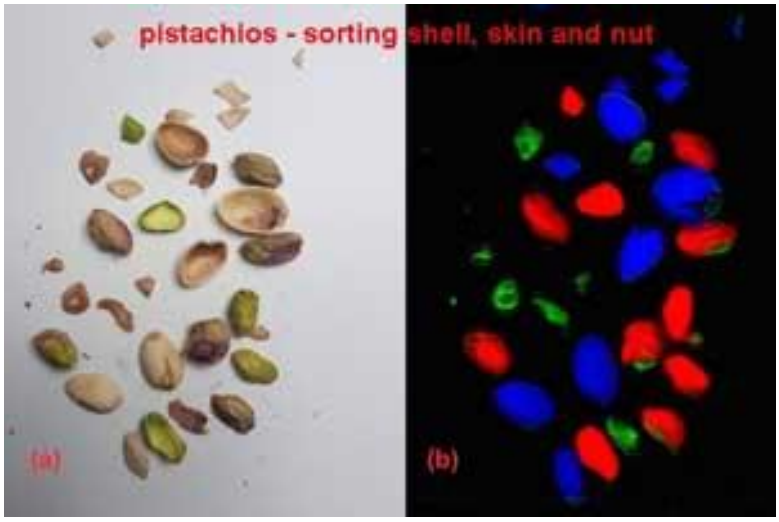


Fig. 1.8 Pistachios - sorting shell, skin and nut

Commercial adoption of hyperspectral sorters is also advancing at a fast pace in the potato processing industry where the technology promises to solve a number of outstanding product quality problems. Work is underway to use hyperspectral imaging to detect "sugar

ends," "hollow heart" and "common scab," conditions that plague potato processors. Fig 1.9 represents Hyperspectral image of "sugar end" potato strips shows invisible defects

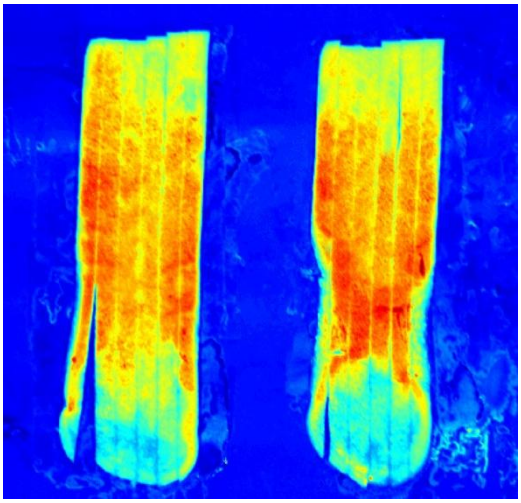


Fig. 1.9 Hyperspectral image of "sugar end" potato strips shows invisible defects

4) MINERALOGY -Geological samples, such as drill cores, can be rapidly mapped for nearly all minerals of commercial interest with hyperspectral imaging. Fusion of **SWIR** and **LWIR** spectral imaging is standard for the detection of minerals in the feldspar, silica, calcite, garnet, and olivine groups, as these minerals have their most distinctive and strongest spectral signature in the LWIR regions. Fig 1.10 shows a set of stones is scanned with a Specim LWIR-C imager in the thermal infrared range from 7.7 μm to 12.4 μm. The quartz and feldspar spectra are clearly recognizable.

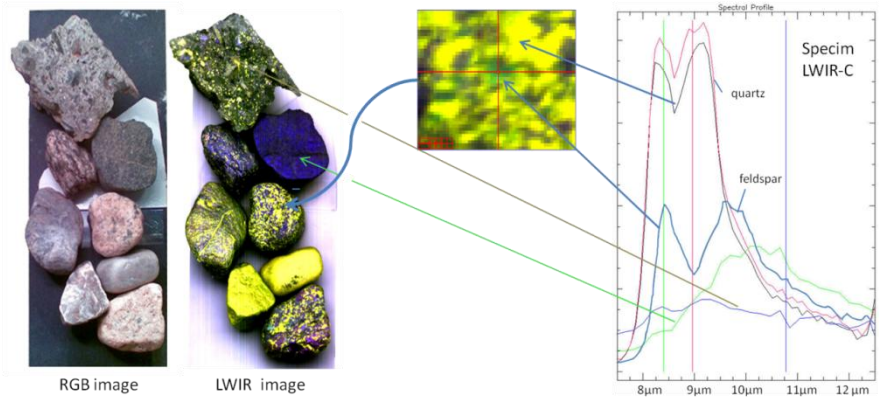


Fig. 1.10 A set of stones is scanned with a Specim LWIR-C imager

Hyperspectral remote sensing of minerals is well developed. Many minerals can be identified from airborne images, and their relation to the presence of valuable minerals, such as gold and diamonds, is well understood. Currently, progress is towards understanding the relationship between oil and gas leakages from pipelines and natural wells, and their effects on the vegetation and the spectral signatures. Recent work includes the PhD dissertations of Werff and Noomen.

5) SURVEILLANCE -Hyperspectral surveillance is the implementation of hyperspectral scanning technology for surveillance purposes. Hyperspectral imaging is particularly useful in **military surveillance** because of countermeasures that military entities now take to avoid airborne surveillance. Aerial surveillance was used by French soldiers using tethered balloons to spy on troop movements during the French Revolutionary Wars, and since that time, soldiers have learned not only to hide from the naked eye, but also to mask their heat signatures to blend into the surroundings and avoid infrared scanning. The idea that drives hyperspectral surveillance is that hyperspectral scanning draws information from such a large portion of the light spectrum that any given object should have a unique spectral signature in at least a few of the many bands that are scanned. The SEALs from NSWDG who killed Osama bin Laden in May 2011 used this technology while conducting the raid (Operation Neptune's Spear) on Osama bin Laden's compound in Abbottabad, Pakistan.

Traditionally, commercially available thermal infrared hyperspectral imaging systems have needed liquid nitrogen or helium cooling, which has made them impractical for most surveillance applications. In 2010, Specim introduced a thermal infrared hyperspectral camera that can be used for outdoor surveillance and UAV applications without an external light source such as the sun or the moon.

Fig 1.11 represents the Hyperspectral thermal infrared emission measurement, an outdoor scan in winter conditions, ambient temperature -15°C—relative radiance spectra from various targets in the image are shown with arrows. The infrared spectra of the different objects such as the watch glass have clearly distinctive characteristics. The contrast level

indicates the temperature of the object. This image was produced with a Specim LWIR hyperspectral imager.

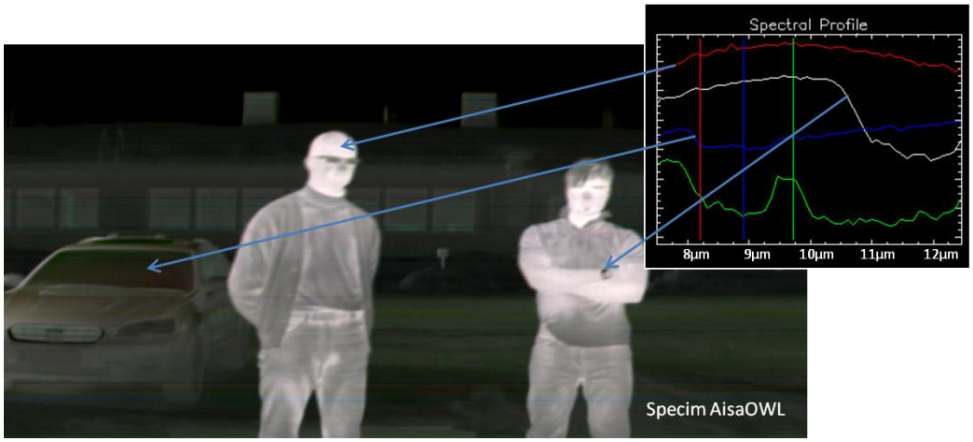


Fig. 1.11 Hyperspectral thermal infrared emission measurement, an outdoor scan in winter conditions

6) REMOTE SENSING -In remote sensing technology it is important to distinguish earth surface features, each features have different spectrum band. Multi spectral satellite can capture image up few bands for example Landsat 7 have 8 bands. But multi spectral imaging satellite can capture earth surface in more than 200 bands which helps scientist to differentiate objects that were not possible in multi spectral imaging because of spectral resolution. Remote sensing is also helpful in detecting the problems inside the agricultural fields, as shown in the Fig 1.12.

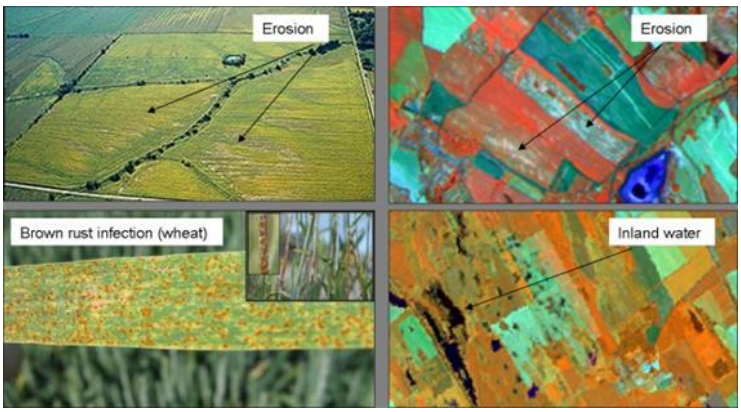


Fig. 1.12 The problems inside the agricultural fields detected using remote sensing

7) ENVIRONMENTAL MONITORING -Hyperspectral imaging is becoming widely popular for tracking changes in the environment. It is commonly used to understand surface CO2 emissions, map hydrological formations, tracking pollution levels, and more.

Most countries require continuous monitoring of emissions produced by coal and oil-fired power plants, municipal and hazardous waste incinerators, cement plants, as well as many other types of industrial sources. This monitoring is usually performed using extractive sampling systems coupled with infrared spectroscopy techniques. Some recent standoff measurements performed allowed the evaluation of the air quality but not many remote independent methods allow for low uncertainty measurements.Fig 1.13 shows the Sub-Aquatic Vegetation Mapping.

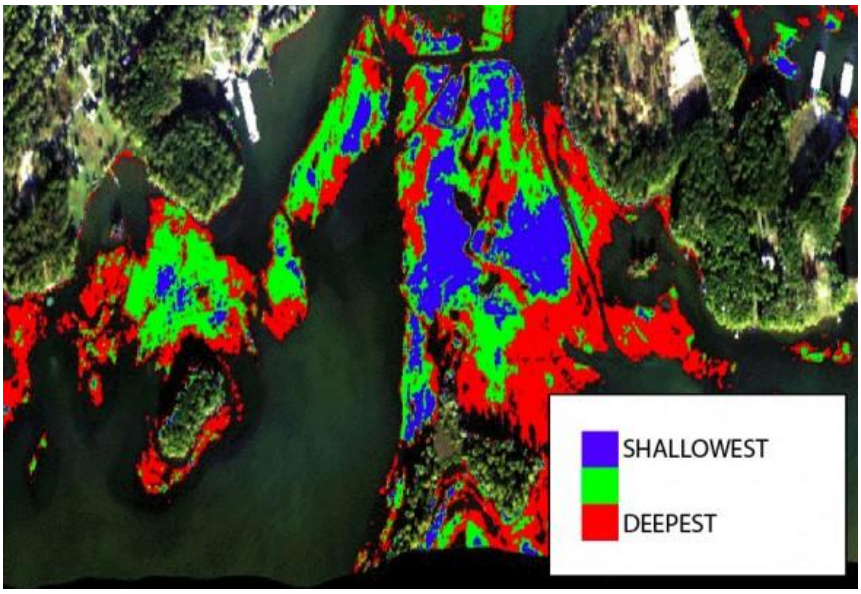


Fig. 1.13 Sub-Aquatic Vegetation Mapping

Hyperspectral imagery can be used to characterize the known absorption features of chlorophyll-a found in phytoplankton and algae. Increases in measured chlorophyll-a content within a water body represent the biological response to nutrient availability and indicator of its trophic state. Such spatially mapped information can be used for point-source and non-point source pollution detection and monitoring. This is represented in the Fig 1.14.

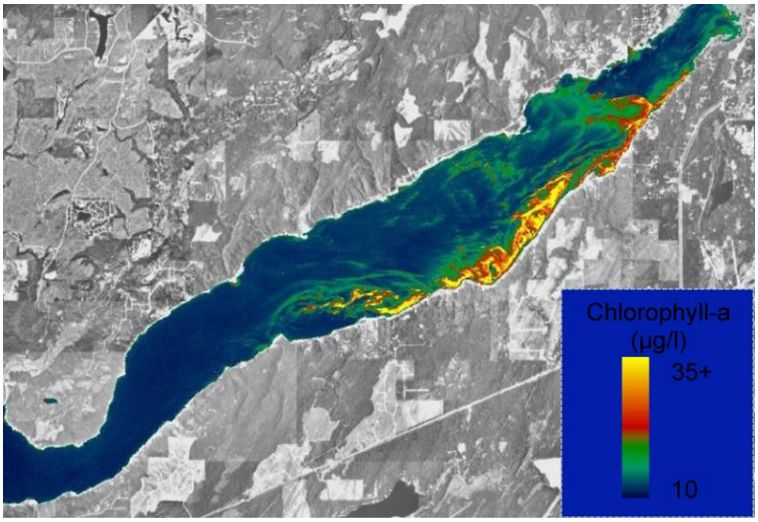


Fig. 1.14 Hyperspectral imagery under water

8) CHEMICAL ENGINEERING -Soldiers can be exposed to a wide variety of chemical hazards. These threats are mostly invisible but detectable by hyperspectral imaging technology. The Telops Hyper-Cam, introduced in 2005, has demonstrated this at distances up to 5 km and with concentrations as low as a few ppm.

Fig 1.15 shows the Remote chemical imaging of a simultaneous release of SF6 and NH3 at 1.5km using the Telops Hyper-Cam imaging spectrometer.

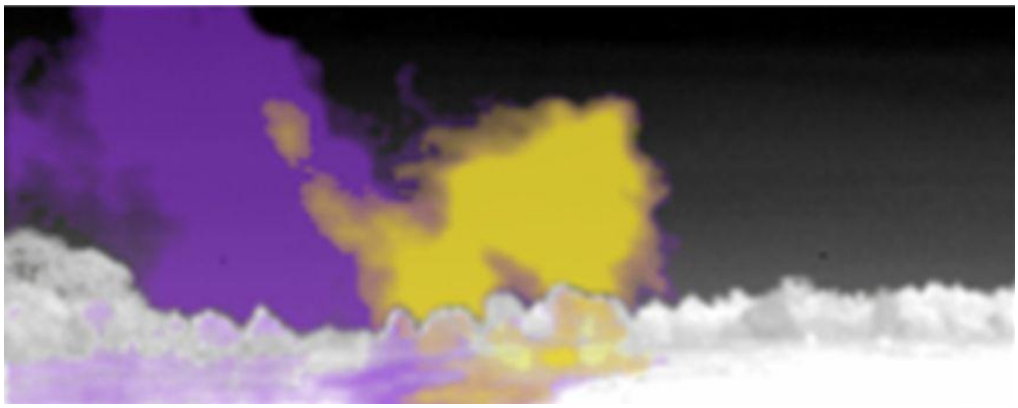


Fig. 1.15 Remote chemical imaging of a simultaneous release of SF6 and NH3

9) FORENSIC SCIENCE -Hyperspectral imaging technology can differentiate fine spectral resolution, which makes suitable in the forensic laboratory. It can be used in different way: questioned document analysis, arson investigation, bloodstain visualization, fibre comparison, gun powder residue, visualization, duct tape examination, fingerprint enhancement, and TLC plate visualization. For example hyperspectral imaging technology can differentiate between dark marks and bloodstains. This type of differentiating is quite important while solving the crime case.

HSI can visualize contrast between bloodstains and dark substrates. Individual droplets can also be imaged in a magnified view to determine the shape of the droplet.

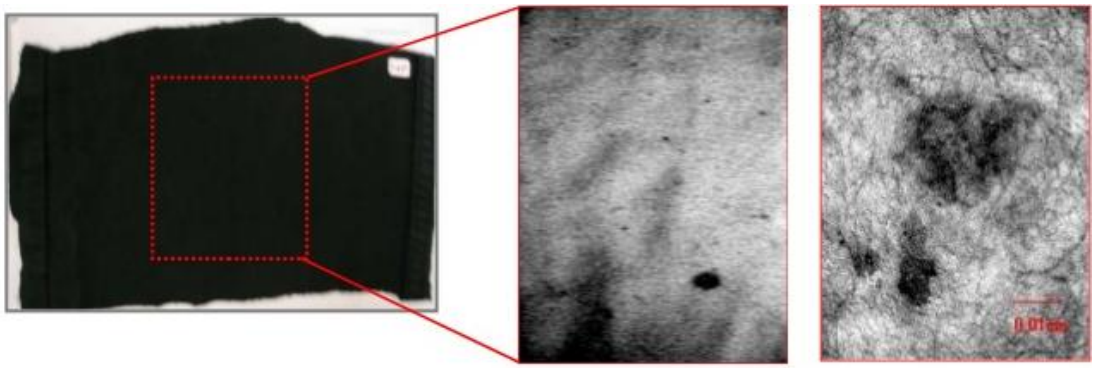


Fig. 1.16 Hyperspectral Imaging for Forensics

In the Fig 1.16 **Left**: Digital image of a piece of black fabric containing bloodstains which cannot be seen with an unaided eye. **Middle**: Hyperspectral images of blood droplets on the black fabric. **Right**: Magnified view of blood droplets on the fabric.

1.7 IMPLEMENTATION

This project work aims at software implementation of CCSDS 123.0-B-1 algorithmin VHDL, analyzing different algorithm parameters with respect to compression parameters and validation of the data. For the software implementation of the algorithm, the entire compression algorithm has been fragmented into multiple modules, with the predictor and encoder being two main entities of the compression standard. An AVIRIS Yellowstone calibrated hyperspectral image is fed as an input to the predictor whose output serves as a

driving input to the encoder. The encoder output is a compressed hyperspectral image. Both the predictor and encoder codes comprise of several software modules in cascade. Finally the entire code has been integrated with the help of a Top Module in VHDL, which receives its input signal drivers from the Top Module Testbench. The VHDL coding has been done in **Libero Version 9.2** which is a product of **Actel Microsemi®**. The software implementation results are verified and compared from the results produced in **Microsoft Visual Studio 2015**. Finally the compression parameters are evaluated for different performance evaluation parameters and the variation in the result is recorded and plotted in MATLAB. The code is provided in Chapter 6 for referring.

1.8 PROBLEM STATEMENT

Hyper spectral images are typically quite large, as they are essentially a compilation of hundreds of images with each containing data for a specific wavelength range. A single AVIRIS pass can generate in excess of 600 Megabytes of data (e.g. AVIRIS images). Problem arises due to the huge amount of hyper spectral data. Hence hyper spectral image compression becomes necessary to reduce the volume of digital data to achieve benefits such as,

a) Reduction of transmission channel bandwidth.

b) Reduction of the buffering and storage requirement.

c) Reduction of data-transmission time at a given rate.

1.9 OBJECTIVE

The aim of the project is to

1. To study CCSDS proposed lossless compression algorithm applied to digital three-dimensional image data from payload instruments, for hyper spectral imagers.
2. Software implementation and testing CCSDS proposed algorithm for prediction and encoding.
3. Implementation of predictor and encoder module in VHDL.

The project involves study, design and simulation of the compression algorithm.

1.10 ORGANIZATION OF THE REPORT

The entire report has been distributed into chapters.

Chapter 1 gives introduction about multispectral and hyperspectral images, covering all aspects of hyperspectral imaging technology, sensors for hyperspectral data collection, data compression and its significance, various data compression techniques, hyperspectral data compression, need for lossless hyperspectral data compression, implementation and modular design of the project, problem statement and aim of the project.

Chapter 2 deals with the literature survey carried out before starting the project. It covers details about our root paper i.e. the CCSDS 123.0-B-1 Lossless Hyperspectral Compression Algorithm Bluebook and other research papers that have moulded our work on this topic and have been of importance during the course of the project.

Chapter 3 provides an overview of CCSDS 123.0-B-1 compression algorithm and also defines the nomenclature used in the report. Details about Consultative Committee for Space Data Systems and its features are covered in this chapter. Lastly, three different types of data reading formats in a hyperspectral image are covered viz. Band Sequential (BSQ), Band Interleaved by Line (BIL) and Band Interleaved by Pixel (BIP). The detailed block diagram of the algorithm is also presented in this chapter.

Chapter 4 deals with Prediction as a method of pre-processing. The different stages of prediction viz. Local Sum, Local Difference, Local Difference Vector, Local Difference Memory, Scaled Prediction, Weight Initialization: Default and Custom, Weight Updation and Mapped Prediction Residual are covered in detail in this chapter. The code snippets written for these modules are also presented for ease of understanding.

Chapter 5 deals with Adaptive Entropy Coder and types of encoding methods involved in it. The different stages of a block adaptive and sample adaptive encoder viz. selecting the best k-option and generating the FS Codeword Sequence are discussed in this chapter.

Chapter 6 consists of the experimental results and simulation waveforms for the multilevel prediction and adaptive encoding strategy.

Chapter 7 provides information regarding conclusion and future work.

CHAPTER 2
LITERATURE SURVEY

CHAPTER 2

LITERATURE SURVEY

The Initial phase of the project included detailed study of multi spectral and hyper spectral images and CCSDS documents related to the data compression. This includes "*CCSDS Recommended Standard for Lossless Data Compression" CCSDS 120 x0g3.* This Standard presents a summary of the key operational concepts like Packet telemetry Data systems, general overview about preprocessor and adaptive encoder. This also addresses system issues relating to the sensor characteristics, error propagation, datapacketization and compression performance [1].

Next phase of the project involved detailed study and analysis of "*Lossless Multispectral & Hyper spectral Image Compression*" *specified in Recommendation for Space Data System Standards, CCSDS 123.0-B-1. Blue Book. Issue 1. Washington, D.C.: CCSDS, May 2012.* This Blue book provides a detailed description of the algorithm for the compression of Hyper Spectral and Multispectral image data and a format for storing the compressed data. The Recommended Standard addresses only lossless compression of three-dimensional data where the requirement is data-rate reduction constrained to allow no distortion to be added in the data compression/decompression process. The Recommended Standard applies to data compression applications of space missions [2].

Survey included study of "*Lossless Data Compression"Recommendation for Space Data System Standards, CCSDS 121.0-B-2. Blue Book. Issue 2. Washington, D.C, CCSDS, May 2012.* This Blue book provides a detailed description of lossless data compression which has been suggested for many space science exploration mission applications. This document also provides a Recommended Standard for a source-coding, data-compression algorithm applied to digital data and to specify how these compressed data shall be inserted into source packets for retrieval and decoding. The recommended standard is applicable to a wide range of

digital data, both imaging and non-imaging and also for the data compression applications of space missions [3].

Document by Emmanuel Christopher presents "*Hyper spectral Data Compression Tradeoff"* provides a detailed description of Data acquisition process, Trends in compression algorithms, describing prediction based algorithms, vector quantization, transform methods and also the tradeoff between compression and the techniques applied for the compression schemes like using lossless compression algorithms that can progressively encode samples, using hybrid solution combining lossy compression algorithms with error encoding techniques [4].

Different hyper spectral and multi spectral image compression algorithm papers were studied namely "Lossless Compression of Hyper spectral Images Based on Searching Optimal Multiband for Prediction" by ChengfuHuo, Rong Zhang, and Tianxiang Peng presents a lossless compression algorithm based on the strength of the correlation between bands. Searching model is constructed using the tree structure. Second, multiband which have strong correlations to each chosen band are found out and are then used to predict the chosen band in a couple-group manner. Lastly, residual images are encoded using entropy coders [5].

The next phase of literature survey was to study adaptive filters and to analyze the adaptive filter mentioned in *CCSDS 123.0-B-1. Blue Book* [2].Implementation and analysis of adaptive filters like LMS algorithm and its variants like sign LMS algorithm, Sign-Sign LMS algorithm etc. werecarried out using the book "*Statistical Digital Signal Processing and Modeling Textbook " by* Monson H. Hayes [6].

Peg Shippert provides detailed description about "Introduction to Hyper spectral Analysis", in which advantages of hyper spectral image is described in detail and also spectral image basics is explained with respect to wave spectrum and also describes how hyper spectral imagery is positioned to become one of the most common research, exploration, and monitoring technologies used in a wide variety of fields[7].

Yuan Liang, Jianping Li, and KeGuo have proposed a novel algorithm for lossless compression of hyper spectral images using hybrid context prediction which provides higher compression ratios with lower complexity and computational cost.

Two different approaches are feasible in the case of hyperspectral image compression. In the first approach, a single band is compressed independently (two-dimensional coding), which only takes advantage of the spatial redundancies among neighboring pixels of a specific band. In the second approach, a band is compressed independently and using previous bands (three-dimensional coding), which also exploits the existing redundancies between bands. Two dimensional coding includes JPEG-LS, CCSDS 121.0 lossless data compressor and three dimensional coding methodologies include Context based Adaptive Lossless Image Coding (CALIC), Lookup Table (LUT), Fast Lossless (FL) for onboard compression of hyperspectral images. The following section compares the merits and limitations of existing prediction based lossless hyperspectral image compression algorithms suitable for onboard use.

2.1 JPEG- LS

The hyperspectral image to be compressed is converted into one or more Two Dimensional (2D) images. A Three Dimensional (3D) image is flattened to form a 2D image which is then compressed. This is illustrated in Fig 2.1. Flattening methods (a) and (f) of Fig 2.1 tend to yield better performance than the others. Predictor used is simple and effective based on Median Edge Detector (MED)[9]. Golomb rice entropy coding is used, assuming that the prediction error follows a two-sided geometric distribution.

MERITS

1. Lesser complexity
2. Compression for JPEG-LS is generally much faster than JPEG 2000 and much better than the original lossless JPEG standard.

DEMERITS

1. JPEG-LS takes no advantage of similarities between spectral bands.

2. The best performing methods of flattening the image imposes the most significant buffer constraints for push-broom imagers which produce samples in BIP or BIL order.
3. Not designed to handle streaking artifacts in images.

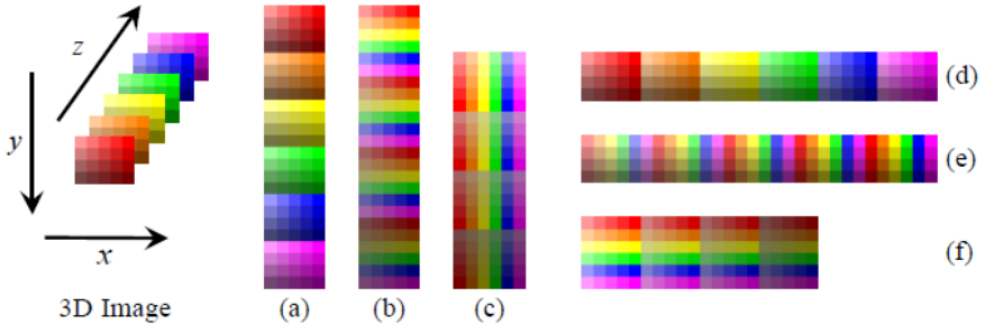


Fig 2.1 Methods of 2D image formation by flattening the samples in a 3D image

2.2 CCSDS 121.0-B-1 ALGORITHM

The Consultative Committee for Space Data Systems (CCSDS) is a multinational forum for the development of communications and data systems standards for spaceflight. The Multispectral & Hyperspectral Data Compression (MHDC) working group of CCSDS develops standards for lossy and lossless data compression, primarily for imagery. The Committee has recommended standard for lossless data compression consisting of a two-dimensional predictor and a block adaptive encoder [3].

The recommendation does not strictly specify the prediction stage, which can be determined by the final user according the specific characteristics of the target data. The subsequent stage consists of a mapper followed by an entropy coder. The entropy coder operates on blocks of J samples. It incorporates multiple coding options, based on Golomb power-of-two codes, which are applied concurrently to a J-samples block. Furthermore, it includes a zero-block and a no compression option, as well as a low entropy option known as second

extension. The algorithm option that yields the shortest encoded length is selected for transmission.

MERITS

1. Very low Computational complexity

DEMERITS

1. Performance decreases significantly with the presence of outliers or when the data do not follow any well-defined statistics.

2.3 OPTIMIZED CONTEXT BASED ADAPTIVE LOSSLESS IMAGE CODING

In optimized Context-based Adaptive Lossless Image Coding (CALIC), the current band is predicted from a reference band which is based on a linear combination of two previous bands. Specifically, the prediction is performed using two samples in the previous bands in the same spatial position of the current sample. Letting Y_λ the current pixel in the current band, $X_{\lambda-1}$ previous band, and $X_{\lambda-2}$ that is next to the previous band. The estimate $\hat{Y}$of the current band is given by equation (2.1).

$$\hat{Y}=\gamma_1+\gamma_2 X_{\lambda-1}+\gamma_3 X_{\lambda-2} \quad (2.1)$$

The coefficients $\gamma_1, \gamma_2, \gamma_3$ are computed by means of an off-line linear regression procedure on training data in such a way so as to minimize the overall energy of the prediction error $|Y_\lambda-\hat{Y}|^2$. This method employs an arithmetic coder in the encoding stage.

MERITS

1. Suitable for compression of data in band interleaved-by-line format
2. Limited amount of onboard memory required.

DEMERITS

1. Multiband CALIC requires a minor complexity increase (about 10%) with respect to 3D-CALIC.
2. Higher Computation Time when compared to JPEG-LS and 3D-CALIC.

2.4 LOOKUP TABLE METHOD

Lookup Table (LUT) method is concerned with searching the previous band for a sample of equal value to the sample co-located to the one to be coded. The sample in the same position as the obtained sample is used as the predictor [9]. First, a pixel value equal to $P_{x,y,z-1}$ is searched. If an equal valued pixel $P_{x',y',z-1}$ is found, then $P_{x,y,z}$ is predicted to have the same value as the pixel in the same position as obtained pixel in the current band $P_{x',y',z}$. Otherwise, the estimated pixel value is equal to the pixel value in the previous band $P_{x,y,z-1}$.

A separate model is adaptively used for the coding of the prediction residual of each band. If the absolute value of the prediction error is between one and a threshold, then the absolute value is encoded using an adaptive range coder. On the other hand, if the absolute value is larger than the threshold, then the threshold is encoded using the same model as before. Also, the absolute value is divided into the least and the most significant bytes. Both of them are separately encoded using a different model for each.

MERITS

1. The time complexity of the LUT method is very low (the prediction requires only two memory operations per pixel a fetch operation from an LUT and a store operation to the same LUT.)
2. Lesser Computational Complexity.

DEMERITS

1. Exploits artifacts that are sometimes introduced by the calibration process, making them less appealing for on-board use, where those artifacts are not likely to occur.
2. Higher memory requirements-The LUT method requires a full LUT for each band.

2.5 FAST LOSSLESS METHOD WITH ARITHMETIC CODING

Fast Lossless (FL) is a 3D coding algorithm that consists of predicting a pixel using only causal information (that part of the image that has already been processed). The FL algorithm predicts the value of each sample considering the values of previously encoded samples in a

small three dimensional neighborhood [5]. This calculation makes use of sample values in the current and P preceding spectral bands. Prediction is accomplished using adaptive linear prediction with prediction weights updated using the sign algorithm.

The entropy coder used to code the prediction residuals stage consists of an arithmetic coder.

MERITS

1. Lesser Memory requirements.
2. Suitable for both calibrated and uncalibrated images.
3. Yields superior compression performance when compared to JPEG-LS, CALIC and LUT.

DEMERITS

1. Entropy coder used is Arithmetic coder which leads to higher computation time.
2. Uses more of real valued arithmetic in prediction stage.

CHAPTER 3

CCSDS 123.0-B-1 ALGORITHM

CHAPTER 3

CCSDS 123.0-B-1 ALGORITHM

3.1 CCSDS

The Consultative Committee for Space Data Systems (CCSDS) is an organization officially established by the management of its members (NASA, ESA, JAXA, etc.). The Committee meets periodically to address data systems problems that are common to all participants, and to formulate sound technical solutions to these problems. Inasmuch as participation in the CCSDS is completely voluntary, the results of Committee actions are termed **Recommended Standards** and are not considered binding on any Agency. This Recommended Standard specifies a method for lossless compression of multispectral and hyperspectral image data and a format for storing the compressed data.

3.2 OVERVIEW

In this section, we describe a block diagram of hyper spectral image compressor and give a brief description of the different blocks. The input to the compressor is an image, which is a three-dimensional array of integer sample values. The compressed image output from the compressor is an encoded bit stream from which the input image can be recovered exactly. Because of the variations in image content, the length of the compressed images will vary from image to image. A Lossless source coding technique preserves source data accuracy and removes redundancy in the data source. In the decoding process, the original data can be reconstructed from the compressed data by restoring the removed redundancy. The decompression process adds no distortion. This technique is particularly useful when data integrity cannot be compromised. The price it pays is generally a lower compression ratio, which is defined as the ratio of the number of original uncompressed bits to the number of compressed bits including overhead bits necessary for signaling parameters. The lossless source coder consists of two separate functional parts namely the predictor and the adaptive entropy coder as shown in Fig 3.1.

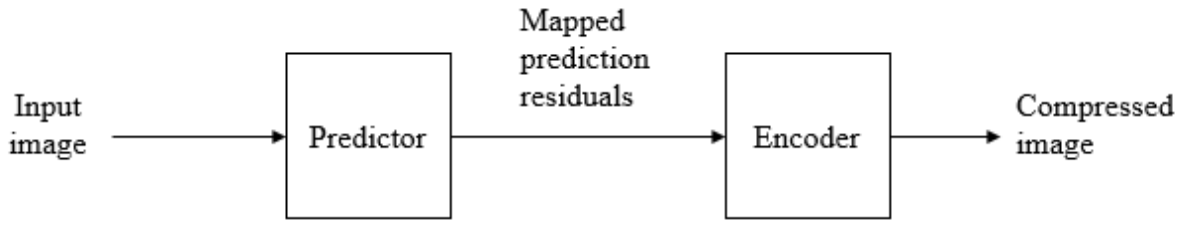


Fig 3.1 Compressor Schematic

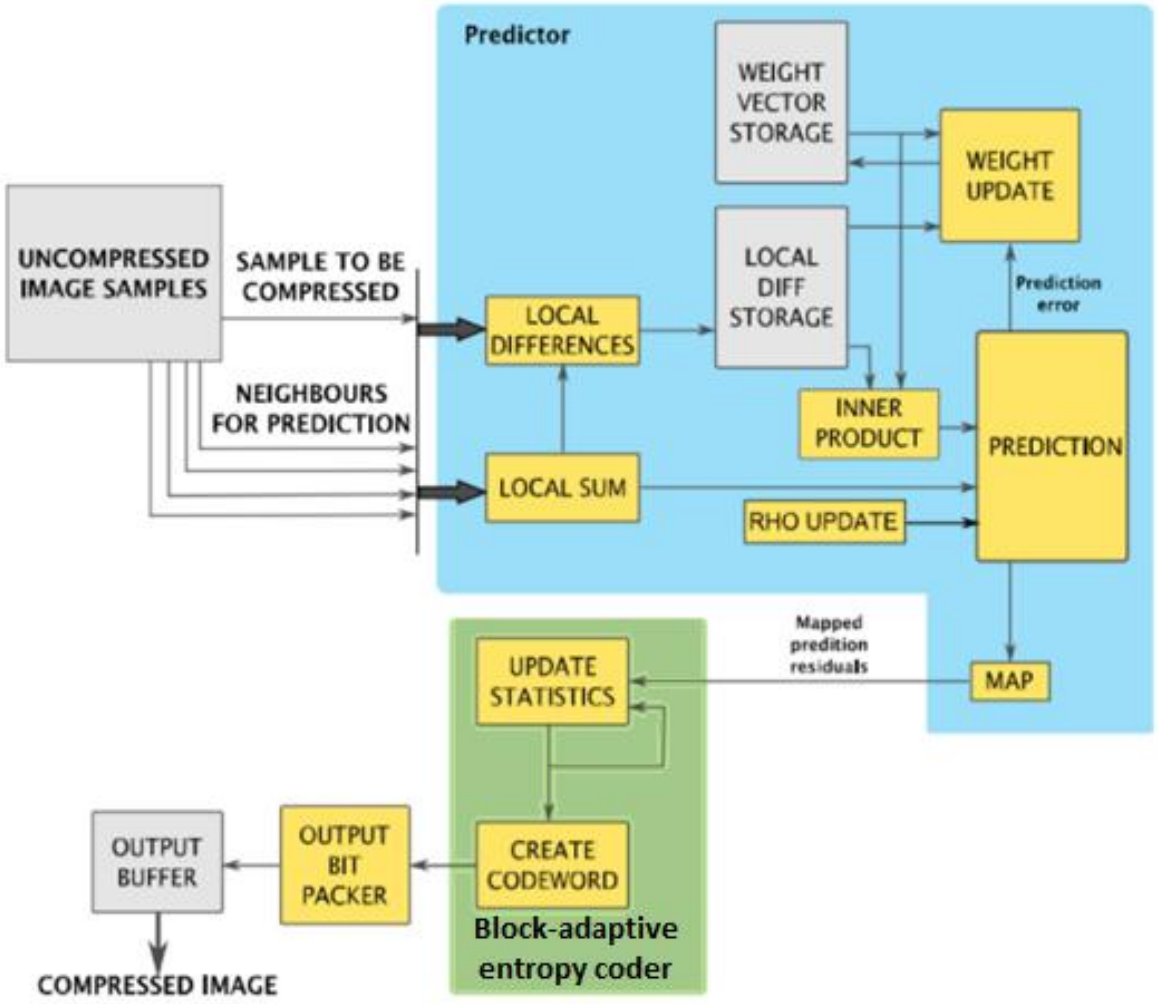


Fig. 3.2 Simplified Schematic of an Implementation of the Recommended Standard

Fig. 3.2 shows the detailed block diagram of the CCSDS 123.0-B-1 compression algorithm. The entire compressor is divided into two parts: Predictor and Encoder. The uncompressed image samples (pixel values) are fed as input to the Predictor, along with the neighbouring

samples for neighbourhood – oriented prediction. The output of local sum is given to the local difference module. Once computed, the local differences are stored in a memory (called RAM, in our case) to be read from later. These central and directional local differences are read one at a time for every sample and put in a column vector called $\boldsymbol{U}_z(t)$. Simultaneously the weights get initialized in the weight initialization module and are stored as a column vector called Weight Vector $\boldsymbol{W}_z(t)$. This vector is updated for every new incoming pixel into the Predictor. Now the inner product of the weight vector and the local difference vector is taken to get the predicted central local difference $(\boldsymbol{W}_z^T(t)\,.\,\boldsymbol{U}_z(t))$. This predicted local difference is fed to the prediction module where the scaled predicted value, predicted sample value, sign of prediction error is calculated. It also receives input from the weight update scaling component (Rho). Finally the prediction residual that is obtained is mapped to a 16 – bit unsigned integer called the Mapped Prediction Residual, which is the output of the predictor.

A block of 8 mapped prediction residual samples from the Predictor are fed to the adaptive entropy coder at a time. The encoder being used here is the Block Adaptive Entropy Coder. There are three modules in the encoder: choosing the best method of encoding and (*k*-option), generating the FS Codeword and transmitting the encoded bits along with the residual bits. So the best possible *k*-option is chosen (the one that corresponds to minimum code word length) and the selection switch selects the *k* split bits. The corresponding encoding option is chosen and the input samples are encoded with the FS Codeword. Once encoded they are transmitted to the ground station.

3.2.1 PREDICTOR

The predictor specified in detail in Chapter 4 uses an adaptive linear prediction method to predict the value of each image sample based on the values of nearby samples in a small three dimensional neighborhood. Prediction is performed sequentially in a single pass. The prediction residual $\delta_{z,y,x}$ i.e., the difference between the predicted and actual sample values, is then mapped to an unsigned integer that can be represented using the same number of bits as the input data sample. These mapped prediction residuals make up the predictor output.

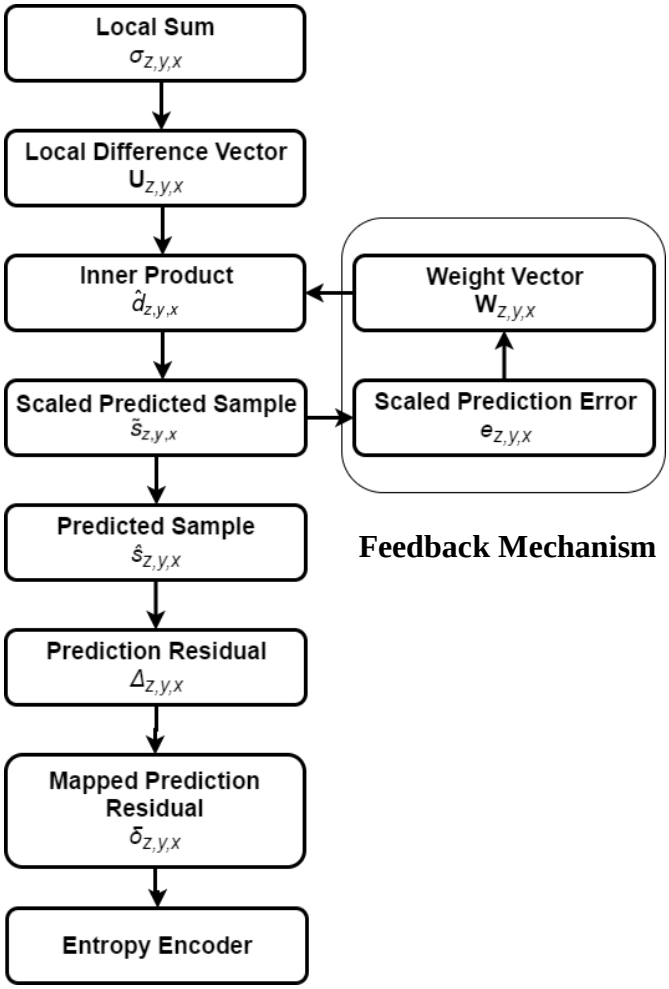


Fig. 3.3 Predictor Design Flowchart

As shown in Fig. 3.3 the entire compression algorithm has been fragmented into multiple modules, with the predictor and encoder being two main entities of the compression standard. The hyperspectral image is fed as an input to the predictor whose output serves as a driving input to the encoder. The encoder output is a compressed hyperspectral image.

3.2.2 ENCODER

Encoder encodes the mapped prediction residuals. Entropy coder parameters are adaptively adjusted during the process to adapt to the changes in the statistics of the mapped prediction residuals. The function of the adaptive entropy coder is to calculate uniquely decipherable, **variable length code words** corresponding to each block of samples input from the

preprocessor. The entropy coder incorporates multiple coding options, each exhibiting efficient performance over different yet overlapping ranges of entropy. The coder selects the coding option that gives the highest compression ratio among the various options on the same block of J samples. A code option "Identifier", requiring only a few bits, is attached before the first code word bit in a coded block to signal the coding option to the decoder for proper decompression. Since the block size J can be small and a new code option is selected for each block, the overall coding can adapt to rapid changes in data statistics.

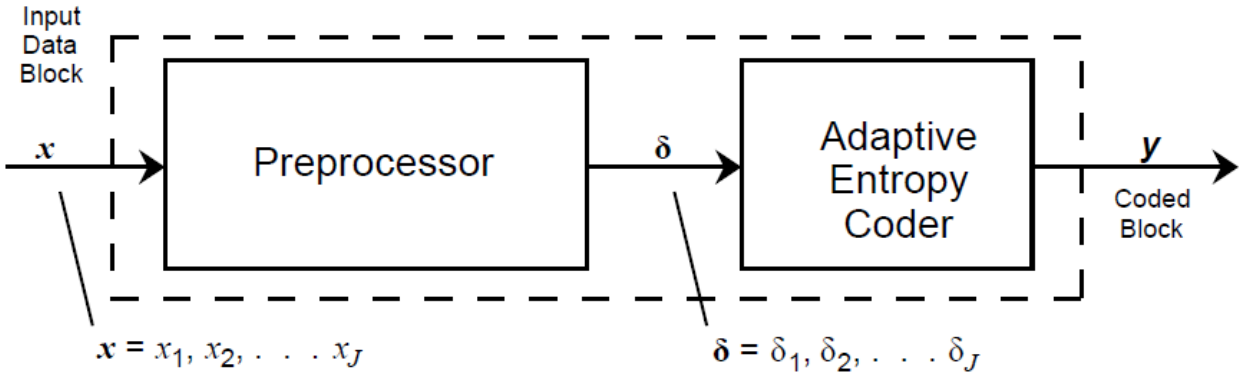


Fig. 3.4 Source Coder

Fig. 3.4 depicts the general representation of the source coder which feeds on an input in the form of a block of data instead of one sample at a time. The adaptive entropy coder being used here is the Block Adaptive Encoder.

3.3 NOMENCLATURE

This section defines parameters and notation pertaining to an image.

3.3.1 DIMENSIONS

An *image* is a three-dimensional array of signed or unsigned integer sample values $S_{z,y,x}$, where x and y are indices in the spatial dimensions, and the index z indicates the spectral band. Indices x, y, and z take on integer values in the ranges $0 \leq x \leq N_X-1$, $0 \leq y \leq N_Y-1$, $0 \leq z \leq N_Z-1$, where each image dimension N_X, N_Y, and N_Z shall have a value of at least 1 and at most 2^{16}.

3.3.2 DYNAMIC RANGE

Data samples shall have a fixed-size dynamic range of *D* bits, where *D* shall be an integer in the range $2 \leq D \leq 16$.

The quantities $s_{\min}$, $s_{\max}$, and s_{mid} denote the lower sample value limit, the upper sample value limit, and a mid-range sample value, respectively. When samples are unsigned integers, the values of $s_{\min}$, $s_{\max}$, , and s_{mid} are defined as

$$s_{\min} = 0,\ s_{\max}, = 2^{D}-1,\ s_{\mathrm{mid}} = 2^{D-1} \tag{3.1}$$

and when samples are signed integers, the values of $s_{\min}$, S_{max}, and S_{mid} are defined as

$$s_{\min} = -2^{D-1},\ s_{\max}, = 2^{D-1}\text{-}1,\ s_{\mathrm{mid}} = 0 \tag{3.2}$$

3.3.3 SAMPLE COORDINATE INDICES

For notational simplicity, data samples and associated quantities may be identified either by reference to the three indices *x*, *y*, *z*, (e.g., $s_{\boldsymbol{z,y,x}}$, $\delta_{\boldsymbol{z,y,x}}$, etc.), or by the pair of indices *t*, *z*, (e.g., $s_{\boldsymbol{z}}(t)$,$\delta_{\boldsymbol{z}}$ (*t*), etc.). That is,

$$s_{\boldsymbol{z}}\ (t) \equiv s_{\boldsymbol{z,y,x}} \tag{3.3}$$

etc., where δ is

$$\delta_{\boldsymbol{z}}(t) \equiv \delta_{\boldsymbol{z,y,x}} \tag{3.4}$$

$$t = y \cdot N_X + x \tag{3.5}$$

Table 3.1 Coordinate Indices and Image Quantities

Symbols	Meaning
x, y, z	Image coordinate indices
T	Alternate image coordinate index
$s_{z,y,x}$,$s_z(t)$	Image data sample
N_X, N_Y, N_Z	Image dimensions (user-specified)
D	Image dynamic range in bits (user-specified)
$s_{\min}$, $s_{\max}$	Lower and upper sample value limits
s_{mid}	Mid-range sample value

Table 3.1 above lists the nomenclature used in the entire project for naming the pixels, the number of rows (4, here), columns (4 here) and bands (5 here), dynamic range (which is 16 in our project). Sample pixel minimum (which is 0 in our case), mid and maximum (which is $2^{16} - 1 = 65535$ in our case) values.

3.4 PIXEL ORDERING TYPES - BIL, BIP AND BSQ

Band Interleaved by Line(BIL), **Band Interleaved by Pixel (**BIP), and **Band Sequential** (BSQ) are three common methods of organizing image data for multiband images. BIL, BIP, and BSQ are not in themselves image formats but are schemes for storing the actual pixel values of an image in a file. These files support the display of single and multiband images and handle black - and - white, grayscale, pseudo color, true color, and multispectral image data. The BIL, BIP, and BSQ files are binary files, and they must have an associated ASCII file header to be interpreted properly by ArcGIS. This header file contains ancillary data about the image such as the number of rows and columns in the image, if there is a color map, and latitude and longitude.

Band interleaved by line data stores pixel information band by band for each line, or row, of the image. For example, given a three - band image, all three bands of data are written for row 1, all three bands of data are written for row 2, and so on, until the total number of rows in the image is reached. The Fig. 3.5 illustrates BIL data for a three - band dataset:

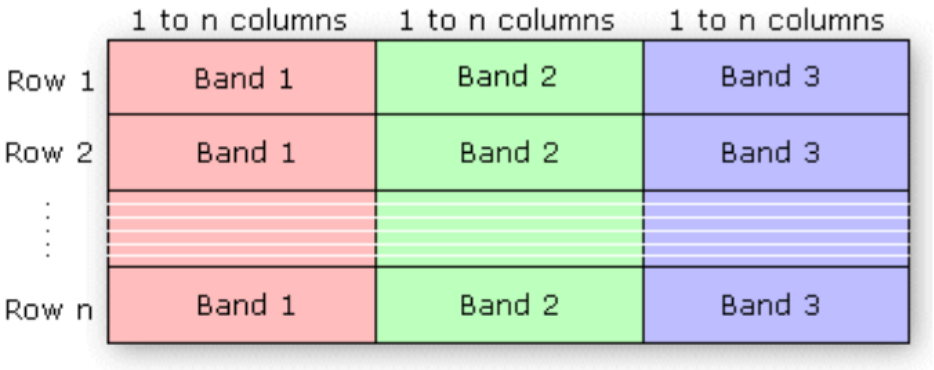


Fig 3.5 Band Interleaved by Line (BIL)

Band interleaved by pixel data is similar to BIL data, except that the data for each pixel is written band by band. For example, with the same three - band image, the data for bands 1, 2, and 3 are written for the first pixel in column 1; the data for bands 1, 2, and 3 are written for the first pixel in column 2; and so on. Fig. 3.6 shows a BIP format representation.

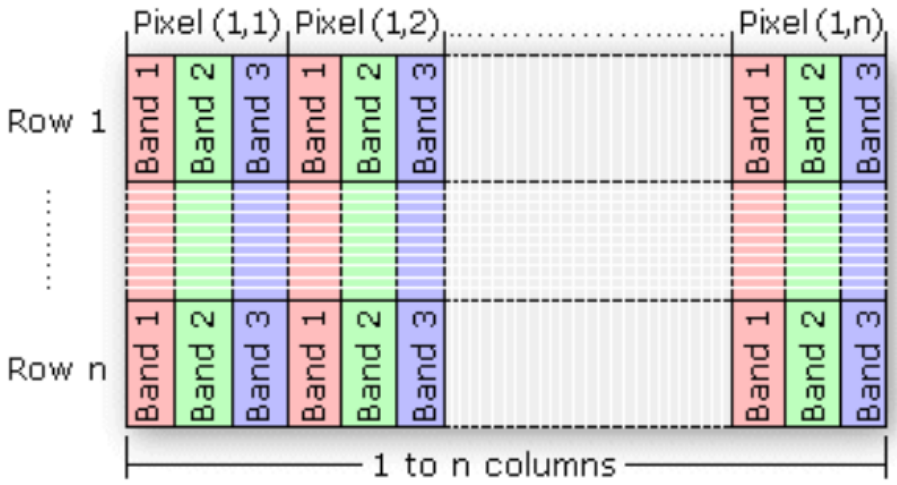


Fig 3.6 Band Interleaved by Pixel (BIP)

Band sequential format stores information for the image one band at a time. In other words, data for all the pixels for band 1 is stored first, then data for all pixels for band 2, and so on. Fig. 3.7 shows a BSQ format arrangement.

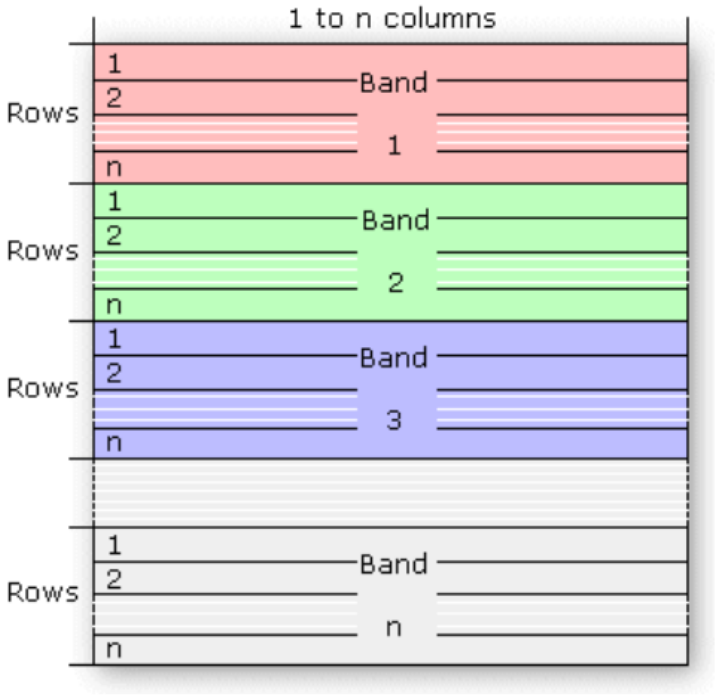


Fig 3.7 Band Sequential (BSQ)

CHAPTER 4
PREDICTOR

CHAPTER 4

PREDICTOR

4.1 OVERVIEW

This section specifies the calculation of the set of predicted sample values $\{\hat{s}_{z,y,z}\}$ and mapped prediction residuals $\{\delta_{z,y,x}\}$ from the input image samples $\{s_{z,y,x}\}$. Quantities defined in this section are summarized in Table 4.1.

Prediction can be performed causally in a single pass through the image. Prediction at samples$s_{z,y,x}$, i.e., the calculation of $\hat{s}_{z,y,z}$and $\delta_{z,y,x}$generally depends on the values of nearby samples in the current spectral band and P preceding (i.e., lower-indexed) spectral bands, where P is a user-specified parameter. Fig. 4.1 illustrates the typical neighborhood of samples used for prediction; this neighborhood is suitably truncated when y = 0, x = 0, x = N_X− 1, or z < P.

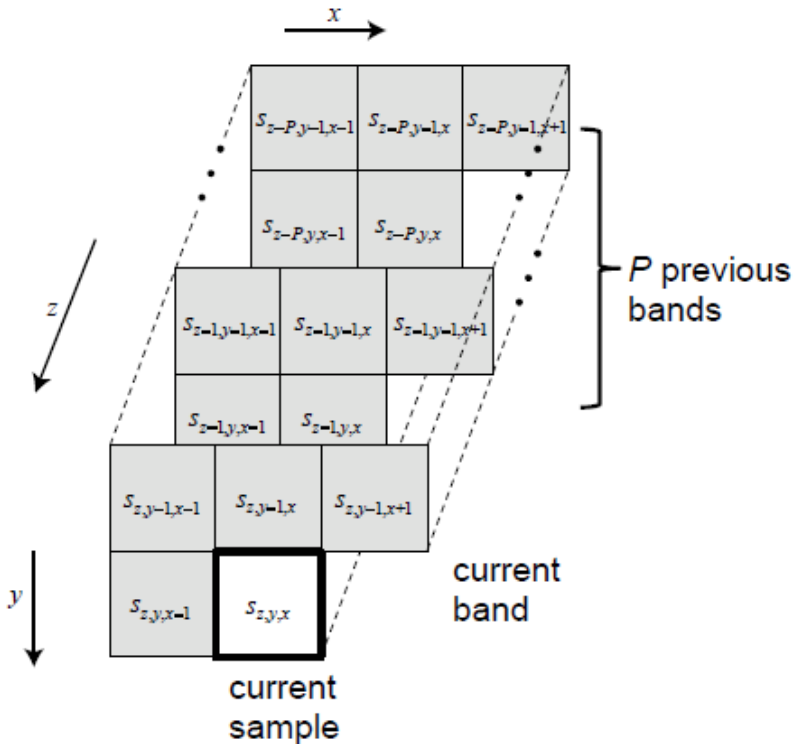
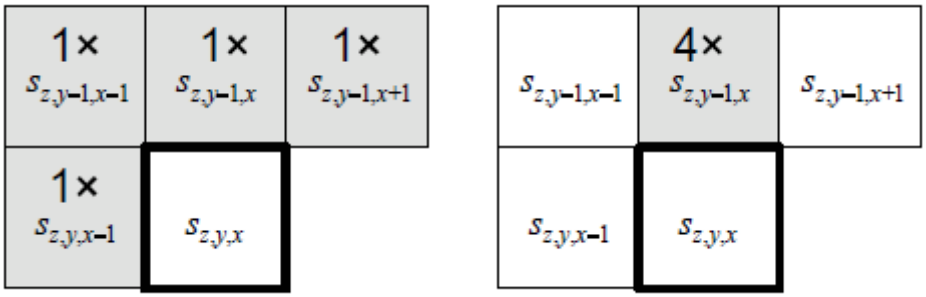


Fig. 4.1 Typical Prediction Neighborhood

Within each spectral band, the predictor computes a *local sum* of neighboring sample values (see 4.4). Each such local sum is used to compute a *local difference* (see 4.5). Predicted sample values are calculated using the local sum in the current spectral band and a weighted sum of local difference values from the current and previous spectral bands (see 4.7). The *weights* (see 4.6) used in this calculation are adaptively updated (see 4.8) following the calculation of each predicted sample value. Each prediction residual, that is, the difference between a given sample value $s_{z,y,x}$ and the corresponding predicted sample value $\hat{s}_{z,y,z}$, is mapped to an unsigned integer $\delta_{z,y,x}$, the mapped prediction residual (see 4.9).

The local sum $\sigma_{z,y,x}$ (see 4.4) is a weighted sum of samples in spectral band z that are adjacent to sample $s_{z,y,x}$. Fig. 4.2 illustrates the samples used to calculate the local sum. A user may choose to perform prediction using ***neighbor-oriented*** or ***column-oriented*** local sums for an image. When neighbor-oriented local sums are used, the local sum is equal to the sum of four neighboring sample values in the spectral band (except when $y = 0$, $x = 0$, or $x = N_X - 1$, in which case these four samples are not all available and the local sum calculation is suitably modified as detailed in 4.4). When column-oriented local sums are used, the local sum is equal to four times the neighboring sample value in the previous row (except when $y = 0$, in which case this sample is not available and the local sum calculation is suitably modified as detailed in 4.4).



Neighbor oriented **Column Oriented**

Fig. 4.2 Samples Used to Calculate Local Sum

The local sums are used to calculate local difference values. In each spectral band, the *central local difference*, $d_{z,y,x}$ is equal to the difference between the local sum $\sigma_{z,y,x}$ and four

times the sample value $s_{z,y,x}$(see 4.5.1). The three *directional local differences*, $d^{N}_{z,y,x}$, $d^{W}_{z,y,x}$, $d^{NW}_{z,y,x}$are each equal to the difference between $\sigma_{z,y,x}$and four times a sample value labeled as 'N', 'W', or 'NW' in Fig. 4.3 (except when this sample value is not available, i.e., at image edges, as detailed in 4.5.2).

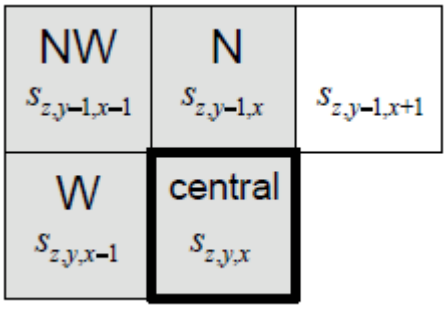


Fig. 4.3 Computing Local Differences in a Spectral Band

A user may choose to perform prediction for an image in *full* or *reduced* mode (see 4.3). Under reduced mode, prediction depends on a weighted sum of the central local differences computed in preceding bands; the directional local differences are not used, and thus need not be calculated, under reduced mode. Under full mode, prediction depends on a weighted sum of the central local differences computed in preceding bands and the three directional local differences computed in the current band.

The use of reduced mode in combination with column oriented local sums tends to yield smaller compressed image data volumes for raw (un-calibrated) input images from push-broom imagers that exhibit significant along-track streaking artifacts. The use of full mode in combination with neighbor-oriented local sums tends to yield smaller compressed image data volumes for whiskbroom imagers, frame imagers, and calibrated imagery. The prediction residual, the difference between the sample value $s_{z,y,x}$and the predicted sample value$\hat{s}_{z,y,z}$, is mapped to an unsigned integer $\delta_{z,y,x}$(see 4.9). This mapping is invertible, so that the decompressor can exactly reconstruct the sample value $s_{z,y,x}$, and has the property that $\delta_{z,y,x}$ can be represented as a *D*-bit unsigned integer.

4.2 NUMBER OF BANDS FOR PREDICTION

The user-specified parameter P, which shall be an integer in the range $0 \leq P \leq 15$, determines the number of preceding spectral bands used for prediction. Specifically, prediction in spectral band z depends on central local differences, defined in 4.5.1, computed in bands z – 1, z – 2…..,z - P_z^*, where

$$P_z^* = \min\{z, P\} \tag{4.1}$$

4.3 FULL AND REDUCED PREDICTION MODES

A user may choose to perform prediction using *full* or *reduced* mode for an image, except when the image has width 1 (i.e., N_X=1), in which case reduced mode shall be used. Under both full and reduced modes, prediction in spectral band z makes use of central local differences from the precedingP_z^*spectral bands. Under full prediction mode, prediction in spectral band z additionally makes use of three directional local differences, defined in 4.5.2, computed in the current spectral band z. Thus the number of local difference values used for prediction at each sample in band z, denoted as C_z, is

$$C_z = \begin{cases} P_z^*, & \text{reduced prediction mode} \\ P_z^* + 3, & \text{full prediction mode} \end{cases} \tag{4.2}$$

The entire compression algorithm has been fragmented into multiple modules, with the predictor and encoder being two main entities of the compression standard. The hyperspectral image is fed as an input to the predictor whose output serves as a driving input to the encoder. The encoder output is a compressed hyperspectral image. In this project, number of **Prediction bands used are 4**.

4.4 LOCAL SUM

The *local sum* $\sigma_{z,y,x}$is an integer equal to a weighted sum of previous sample valuesin band z that are neighbors of sample $s_{z,y,x}$.A user may choose to perform prediction using*column-oriented* or *neighbor-oriented* local sums for an image, except when the image has width 1 (i.e., N_X=1), in which case column-oriented local sums shall be used.When neighbor-oriented

local sums are used, the local sum is equal to the sum of four neighboring sample values in the spectral band. For an image, except when the image has width 1 (i.e., N_X=1), in which case column-oriented local sums shall be used. When column-oriented local sums are used, the local sum is equal to four times the neighboring sample value in the previous row. Column-oriented local sums are not recommended under full prediction mode.

When neighbor-oriented local sums are used, $\sigma_{z,y,x}$ is defined as,

$$\sigma_{z,y,x} = \begin{cases} s_{z,y,x-1} + s_{z,y-1,x-1} + s_{z,y-1,x} + s_{z,y-1,x+1}, & y > 0,\ 0 < x < N_X - 1 \\ 4s_{z,y,x-1}, & y = 0, x > 0 \\ 2\left(s_{z,y-1,x} + s_{z,y-1,x+1}\right), & y > 0, x = 0 \\ s_{z,y,x-1} + s_{z,y-1,x-1} + s_{z,y-1,x}, & y > 0, x = N_X - 1 \end{cases} \quad (4.3)$$

And when column-oriented local sums are used, $\sigma_{z,y,x}$ is defined as

$$\sigma_{z,y,x} = \begin{cases} 4s_{z,y-1,x}, & y > 0 \\ 4s_{z,y,x-1}, & y = 0,\ x > 0 \end{cases} \quad (4.4)$$

All the modules in this project have been implemented using the neighbor orientation. In this project the four equations for neighbor-oriented local sum are realized using a **4:1 MULTIPLEXER**, where the four conditions of the MUX are the four equations in (4.3) and select inputs are two select lines x and y which define the boundary conditions. The equation corresponding to the appropriate pixel location and boundary condition is selected and the local sum for that pixel is calculated. Fig. 4.4 shows the 4:1 MUX.

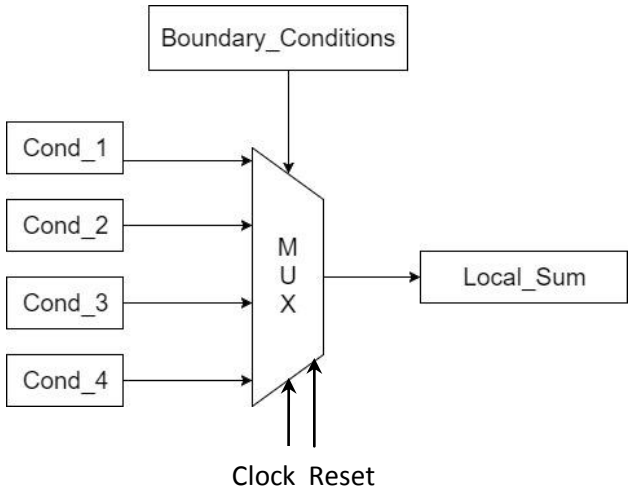


Fig. 4.4 4:1 MUX for Local Sum

The VHDL code for realizing the MUX function has been written in Libero 9.2 as shown below.

```
-- Multiplexer output for Local Sum module
   process(sel,y1,y2,y3,y4) is
 begin

     case sel is
         when "00" => localSum <= y1;
         when "01" => localSum <= y2;
         when "10" => localSum <= y3;
         when "11" => localSum <= y4;
         when others => localSum <= "0000000000000000000";
     end case;

 end process;

-- Output Conditions for Local Sum : MUX Output
     y1 <= ("00" & a) + ("00" & b) + ("00" & c) + ("00" & d);
     y2 <= ((a(15 downto 0) & "00"));
     z1 <= ("00" & c) + ("00" & d);
     y3 <= (z1(16 downto 0) & '0');
     y4 <= ("00" & a) + ("00" & b) + ("00" & c) + ("00" & c);
```

The output of local sum is given to the input of local difference module using PORT MAPPING.

4.5 LOCAL DIFFERENCES

Local Difference of any sample is the difference between the current sample and local sum value described in the previous section. There are two types of local differences calculated. They are called Central Local Difference and Directional Local Differences.

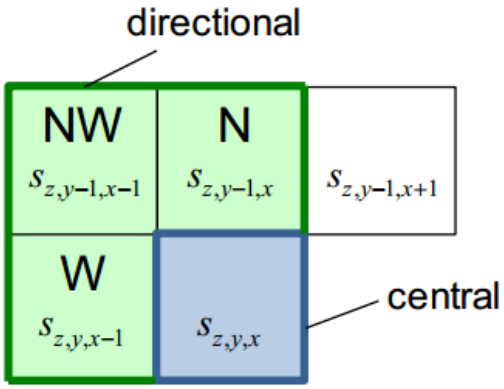


Fig. 4.5 Minuends for Computing Directional and Central Local Differences in a Spectral Band

Fig. 4.5 shows the minuends for computing directional and central local differences in a spectral band. The immediate left pixel is the West pixel, the immediate top pixel is the North pixel and the diagonally left one is the North-West pixel. The pixel under consideration is the central pixel.

4.5.1 CENTRAL LOCAL DIFFERENCE

In each Spectral band, Central local difference $d_{z,y,x}$ is equal to the difference between the local sum $\sigma_{z,y,x}$ and four times the sample value $s_{\boldsymbol{z,y,x}}$. When *x* and *y* are not both zero (i.e., when *t* > 0), the central local difference $d_{z,y,x}$ is defined as

$$d_{z,y,x} = 4s_{\boldsymbol{z,y,x}} - \sigma_{z,y,x} \tag{4.5}$$

4.5.2 DIRECTIONAL LOCAL DIFFERENCES

The three directional local differences $d^N_{z,y,x}$, $d^W_{z,y,x}$, $d^{NW}_{z,y,x}$ are each equal to the differencebetween$\sigma_{z,y,x}$ and four times a sample value labeled as'N', 'W', or 'NW'. User may choose to perform prediction for an image in full or reduced mode as described in section 4.3.

When *x* and *y* are not both zero (i.e., when *t* > 0), the three directional local differences are defined as

$$d^N_{z,y,x} = \begin{cases} 4s_{z,y-1,x} - \sigma_{z,y,x}, & y > 0 \\ 0, & y = 0 \end{cases} \tag{4.6}$$

$$d^W_{z,y,x} = \begin{cases} 4s_{z,y,x-1} - \sigma_{z,y,x}, & x > 0, y > 0 \\ 4s_{z,y-1,x} - \sigma_{z,y,x}, & x = 0, y > 0 \\ 0, & y = 0 \end{cases} \tag{4.7}$$

$$d^{NW}_{z,y,x} = \begin{cases} 4s_{z,y-1,x-1} - \sigma_{z,y,x}, & x > 0, y > 0 \\ 4s_{z,y-1,x} - \sigma_{z,y,x}, & x = 0, y > 0 \\ 0, & y = 0 \end{cases} \tag{4.8}$$

Under reduced mode, prediction depends on a weighted sum of the central local differences computed in preceding bands; the directional local differences are not used, and thus need

not be calculated. Under full mode, prediction depends on a weighted sum of the central local differences computed in preceding bands and the three directional local differences computed in the current band.

4.5.3 LOCAL DIFFERENCE VECTOR

For $t > 0$, the local difference vector$\mathrm{U}_z(t)$is a vector of the $C_z$local difference values used tocalculate the predicted sample value $\hat{s}_z(t)$. Under full prediction mode, $\mathrm{U}_z(t)$is defined as

$$\mathrm{U}_z(t) = \begin{bmatrix} d_z^N(t) \\ d_z^W(t) \\ d_z^{NW}(t) \\ d_{z-1}(t) \\ d_{z-2}(t) \\ \vdots \\ d_{z-P_z^*}(t) \end{bmatrix}, \tag{4.9}$$

and under reduced prediction mode, for $z > 0$, $\mathrm{U}_z(t)$is defined as

$$\mathrm{U}_z(t) = \begin{bmatrix} d_{z-1}(t) \\ d_{z-2}(t) \\ \vdots \\ d_{z-P_z^*}(t) \end{bmatrix} \tag{4.10}$$

In order to store the local differences in the vector $\mathbf{U}_z(t)$, all the local difference values for the 4 prediction bands have to be saved in a memory and then read from it serially. For this purpose a storage space that is **8x8** in dimension has been created which serves as a memory with the storage capacity of 64 local difference values. Since we are using **4 rows, 4 columns and 4 prediction bands** in our project, hence **4*4*4 = 64** values need to be saved in the memory entirely. In the VHDL code written for this module the memory has been declared as RAM and is realized as follows:

```
-- Writing the Central Local Differences into the memory
P0: process(POR,WClk)
    begin
        if POR = '0' then
            RAM <= (others => "0000000000000000000");
            WrAddr  <= (others => '0');
        elsif WClk'event and WClk = '1' then
            if WEn = '1' then
               WrAddr <= WrAddr+1;
            end if;
            if WEn = '1' then
                RAM(conv_integer(WrAddr)) <= central_LD;
            end if;
        end if;
    end process;


-- Reading the Central Local Differences from the memory
P1: process(POR,RClk)
    begin
        if POR = '0' then
            iLDVector <= (others => "0000000000000000000");
            REn_L <= '0';

        elsif RClk'event and RClk = '1' then
            Ren_L <= REn;
            if Ren = '1' then
                iLDVector(0) <= RAM(conv_integer(RdAddr));
                if RdAddr(4) = '1' or RdAddr(5) = '1' then
                    iLDVector(1) <= RAM(conv_integer(RdAddr)-16);
                end if;
                if RdAddr(5) = '1' then
                    iLDVector(2) <= RAM(conv_integer(RdAddr)-32);
                end if;
                if RdAddr(5 downto 4) = "11" then
                    iLDVector(3) <= RAM(conv_integer(RdAddr)-48);
                end if;
            end if;
        end if;
    end process;
```

Process P0 and P1 depict writing local differences into the memory and then reading from it serially to be stored in a vector.

4.6 WEIGHTS

4.6.1 WEIGHT VALUES AND WEIGHT RESOLUTION

In the prediction calculation (see 4.7), for $t > 0$ each component of the local difference vector $U_z(t)$is multiplied by a corresponding integer *weight value*. The resolution of the weight values is controlled by the user-specified parameter Ω, which shall be an integer in the range $4 \leq \Omega \leq 19$. Each weight value is a signed integer quantity that can be represented using Ω+3

bits. Thus each weight value has minimum and maximum possible values ω_{min} and ω_{max}, respectively, where

$$\omega_{min} = -2^{\Omega+2}, \ \omega_{max} = 2^{\Omega+2} - 1 \qquad (4.11)$$

4.6.2 WEIGHT VECTOR

The weight vector $W_z(t)$ is a vector of the C_z weight values used in prediction. Under full prediction mode,

$$W_z(t) = \begin{bmatrix} \omega_z^N(t) \\ \omega_z^W(t) \\ \omega_z^{NW}(t) \\ \omega_z^{(1)}(t) \\ \omega_z^{(2)}(t) \\ \vdots \\ \omega_z^{(P_z^*)}(t) \end{bmatrix}, \qquad (4.12)$$

and under reduced prediction mode, for $z > 0$,

$$W_z(t) = \begin{bmatrix} \omega_z^{(1)}(t) \\ \omega_z^{(2)}(t) \\ \vdots \\ \omega_z^{(P_z^*)}(t) \end{bmatrix} \qquad (4.13)$$

where the weight values are calculated as specified in 4.6.3 and 4.8.

4.6.3 WEIGHT INITIALIZATION

A user may choose to use either *default* or *custom* weight initialization, defined below, to select the initial weight vector $W_z(1)$ for each spectral band z. The same weight initialization method shall be used for all spectral bands.

Default Weight Initialization:

When default weight initialization is used, for each spectral band z, initial weight vector components $\omega_z^{(1)}(1)$, $\omega_z^{(2)}(1)$, …, $\omega_z^{(P_z^*)}(t)$, shall be assigned values

$$\omega_z^{(1)}(\mathrm{l})=\frac{7}{8}2^{\Omega},\ \omega_z^{(i)}(\mathrm{l})=\left\lfloor\frac{1}{8}\omega_z^{(i-1)}(\mathrm{l})\right\rfloor,\ i=2,3,\ldots,P_z^*. \quad (4.14)$$

With this option, under full prediction mode the remaining components of $W_z(\mathrm{l})$ shall be assigned values

$$\omega_z^N(\mathrm{l})=\omega_z^W(\mathrm{l})=\omega_z^{NW}(\mathrm{l})=0 \quad (4.15)$$

Custom Weight Initialization:

When custom weight initialization is used, for each spectral band z, the initial weight vector $W_z(\mathrm{l})$ shall be assigned using a user-specified *weight initialization vector* Λ_z, consisting of C_z signed Q-bit integer components.

The weight initialization resolution Q shall be a user-specified integer in the range $3 \le Q \le \Omega+3$ bits. The initial weight vector $W_z(\mathrm{l})$ shall be calculated from Λ_z by

$$W_z(\mathrm{l})=2^{\Omega+3-Q}\Lambda_z+\lceil 2^{\Omega+2-Q}-1\rceil \mathbf{1} \quad (4.16)$$

Where **1** denotes a vector of all 'ones'.

In the $(\Omega+3)$-bit two's complement representation of each component of $W_z(\mathrm{l})$, the Q most significant bits are equal to the binary representation of the corresponding component of Λ_z. The remaining bits, if any, are made up of a '0' bit followed by '1' bits in the remaining positions.

4.7 PREDICTION CALCULATION

Scaled prediction is the next module wherein a **CLIP** function is used to calculate the Scaled Predicted sample value. The input drivers for this module are: local sum, input sample value, minimum and maximum input sample value, and the predicted central local difference. The scaled predicted sample value $\tilde{s}_z(t)$ is an integer defined as

$$\tilde{s}_z(t)=\begin{cases}\mathrm{clip}\left(\left\lfloor\frac{\mathrm{mod}_R^*\left[\hat{\boldsymbol{d}}_{\boldsymbol{z}}(\boldsymbol{t})+2^{\Omega}(\sigma_z(t)-4s_{\mathrm{mid}})\right]}{2^{\Omega+1}}\right\rfloor+2s_{\mathrm{mid}}+1,\{2s_{\mathrm{min}},2s_{\mathrm{max}}+1\}\right), & t>0\\ 2s_{z-1}(t), & t=0,P>0,z>0\\ 2s_{\mathrm{mid}}, & t=0 \text{ and } (P=0 \text{ or } z=0)\end{cases} \quad (4.17)$$

In this calculation:

a) For $t > 0$ the predicted central local difference $\hat{d}_z(t)$is equal to the **inner product** of vectors $W_z(t)$ and $U_z(t)$:

$$\hat{\boldsymbol{d}}_{\boldsymbol{z}}(t) = \mathbf{W}_z^{\mathrm{T}}(t)\mathbf{U}_z(t) \quad (4.18)$$

except for z=0 under reduced mode, in which case$\hat{d}_z(t)$= 0.

b) The user-selected register size parameter R shall be an integer in the range max $\{32, D + \Omega \leq R \leq 64$. Increasing the register size R reduces the chance of an overflow occurring in the calculation of a scaled predicted sample value.

Once the scaled predicted value is obtained it is halved using the **FLOORING** function to obtain the predicted sample value as depicted in eqn. (4.19). The predicted sample value $\hat{s}_z(t)$is defined as

$$\hat{s}_z(t) = \left\lfloor \frac{\tilde{s}_z(t)}{2} \right\rfloor \quad (4.19)$$

In VHDL the code snippet for the above equation is as follows:-

```
-- Calculation of Predicted Sample Value
s_cap <= scaled_predicted_value (16 downto 1);
```

where scaled predicted value is a 17 – bit signed integer.

4.8 WEIGHT UPDATE

Weight updation is an important aspect of prediction because it is aimed at reducing the prediction errors. Weights are initialized with the same values for the first pixel of every band and then updated as newer pixels start coming. Obtaining the scaled prediction error is essential to update weights. The scaled prediction error $e_z(t)$ is an integer defined as

$$e_z(t) = 2s_z(t) - \tilde{s}_z(t) \quad (4.20)$$

For $t > 0$, the **weight update scaling exponent** $\rho(t)$ is an integer defined as

$$\rho(t) = \text{clip}\left(v_{\min} + \left\lfloor \frac{t - N_X}{t_{\text{inc}}} \right\rfloor, \ \{v_{\min}, \ v_{\max}\}\right) + D - \Omega, \qquad (4.21)$$

where user-specified integer parameters $v_{\min}$, $v_{\max}$, and t_{inc} are constrained as follows:

a) The values of $v_{\min}$ and $v_{\max}$ shall be integers in the range $-6 \leq v_{\min} \leq v_{\max}, \leq 9$.

b) The weight update factor change interval t_{inc} shall be a power of 2 in the range $2^4 \leq t_{\text{inc}} \leq 2^{11}$

These parameters control the rate at which weights adapt to image data statistics. The initial weight update scaling exponent is $\rho(1) = v_{\min} + D - \Omega$ and at regular intervals determined by the value of t_{inc}, $\rho(t)$ is incremented by one until reaching a final value $v_{\max} + D - \Omega$. Smaller values of $\rho(t)$ produce larger weight increments, yielding faster adaptation to source statistics but worse steady-state compression performance.

For $t > 0$, following the calculation of $\tilde{s}_z(t)$, the next weight vector in the spectral band, $\mathrm{W_z}(t+1)$, is defined as

$$\boxed{\mathbf{W}_z(t+1) = \text{clip}\left(\mathbf{W}_z(t) + \left\lfloor \frac{1}{2}\left(\text{sgn}^+[e_z(t)] \cdot 2^{-\rho(t)} \cdot \mathbf{U}_z(t) + 1\right)\right\rfloor, \ \{\omega_{\min}, \omega_{\max}\}\right)} \quad (4.22)$$

where the floor and clip operations are applied to each component of the vector. The quantity $\left\lfloor \frac{1}{2}\left(\text{sgn}^+[e_z(t)] \cdot 2^{-\rho(t)} \cdot \mathrm{U}_z(t) + 1\right)\right\rfloor$ is equivalent to $\frac{1}{2}\left(\lfloor \text{sgn}^+[e_z(t)] \cdot 2^{-\rho(t)} \cdot \mathrm{U}_z(t) + 1\right)$ but is not in general equivalent to $\left\lfloor \frac{1}{2}\left(\text{sgn}^+[e_z(t)] \cdot \lfloor 2^{-\rho(t)} \cdot \mathrm{U}_z(t) \rfloor + 1\right)\right\rfloor$.

The VHDL code for the above equation is written using temporary signals to hold the intermediate values and then using a CLIP function to clip the weight values to a maximum and minimum limit. The code for generating the weight vector for next spectral band is:

```
-- Error Calculation
Ez_t <= signed ('0'  & (Pnx(15 downto 0) & '0')) - signed (scaled_predicted_value);

-- Wz_(t+1)
G0: for i in  0 to 6 generate
        temp_1(i) <= Uz_t(i)(18 downto 0) & "000000000";

            process(Ez_t, temp_1(i))
            begin
                if (Ez_t(15) = '0') then
                    temp_2(i) <= (temp_1(i)) + ("00000000000000000000000000001");
                elsif (Ez_t(15) = '1') then
                    temp_2(i) <= (("00000000000000000000000000000" - temp_1(i))
                     + ("00000000000000000000000000001"));
                end if;
            end process;

        temp_3(i) <= temp_2(i)(27) & temp_2(i)(27) & temp_2(i)(27) & temp_2(i)(27)
                     & temp_2(i)(27) & temp_2(i)(27 downto 1);
        temp_4(i) <= Wz_t(i)(21) & Wz_t(i)(21) & Wz_t(i)(21) & Wz_t(i)(21) &
                     Wz_t(i)(21) & Wz_t(i)(21) & Wz_t(i)(21) & Wz_t(i)(21) &
                     Wz_t(i)(21) & Wz_t(i)(21) & Wz_t(i);
        temp_5(i) <= temp_3(i) + temp_4(i);
    end generate;
```

4.9 MAPPED PREDICTION RESIDUAL

The mapped prediction residual is a 16-bit unsigned integer which is fed to the Block Adaptive Encoder. The mapped prediction residual $\delta_z(t)$ is an integer defined as

$$\delta_z(t) = \begin{cases} |\Delta_z(t)| + \theta_z(t), & |\Delta_z(t)| > \theta_z(t) \\ 2|\Delta_z(t)|, & 0 \le (-1)^{\tilde{s}_z(t)}\Delta_z(t) \le \theta_z(t) \\ 2|\Delta_z(t)| - 1, & \text{otherwise} \end{cases} \quad (4.23)$$

Where the prediction residual $\Delta_z(t)$is the difference between the predicted and actual sample values,

$$\Delta_z(t) = s_z(t) - \hat{s}_z(t) \quad (4.24)$$

and $\theta_z(t)$ is defined as

$$\theta_z(t) = \min\{\hat{s}_z(t) - s_{\min},\ s_{\max} - \hat{s}_z(t)\} \quad (4.25)$$

Each mapped prediction residual $\delta_z(t)$ can be represented as a *D*-bit unsigned integer. This is the final output of the predictor. Quantities defined in this section are summarized in Table 4.1.

Table 4.1 Symbols and their meanings

Symbol	Meaning
P	Number of Spectral Bands used for Prediction (user-specified)
P_z^*	Number of Previous Spectral Bands used for Prediction in Band z
C_z	Number of Local Difference values used for Prediction in Band z
$\sigma_{z,y,x}$	Local Sum
$d_{z,y,x}$, $d_z(t)$	Central Local Difference
$d_{z,y,x}^N$, $d_{z,y,x}^W$, $d_{z,y,x}^{NW}$,$d_z^N(t)$, $d_z^W(t)$, $d_z^{NW}(t)$	Directional Local Differences
$\mathrm{U}_z\ (t)$	Local Difference Vector
Ω	Weight Resolution (user-specified)
$\omega_{\min}$, $\omega_{\max}$	Minimum and Maximum Weight Values
$\mathrm{W}_z\ (t)$	Weight Vector
$\omega_z^{\mathrm{N}}\ (t)$, $\omega_z^{\mathrm{W}}\ (t)$, $\omega_z^{\mathrm{NW}}\ (t)$, $\omega_z^{(i)}\ (t)$	Weight Values
Λ_{z}	Weight Initialization Vector (optional, user-specified)
Q	Initial Weight Value Resolution (optional, user-specified)
$\tilde{s}_z\ (t)$	Scaled Predicted Sample Value
$\hat{d}_z\ (t)$	Predicted Central Local Difference
R	Register Size, In-Bits, used in prediction calculation (user-specified)
$\hat{s}_z(t)$, $\hat{s}_{z,y,z}$	Predicted Sample Value
$e_z\ (t)$	Scaled Prediction Error
$\rho(t)$	Weight Update Scaling Exponent
$v_{\min}$, $v_{\max}$	Initial and Final Weight Update Scaling Exponent Parameters (user-specified)
t_{inc}	Weight Update Scaling Exponent Change Interval (user-specified)
$\delta_{\mathrm{z}}\ (t)$, $\delta_{\mathrm{z,y,x}}$	Mapped Prediction Residual
$\Delta_z\ (t)$	Prediction Residual
$\theta_z\ (t)$	Difference between $\hat{s}_z(t)$ and nearest endpoint $s_{\min}$, $s_{\max}$

CHAPTER 5
ENCODER

CHAPTER 5

ENCODER

This chapter specifies the encoding stage of the compressor and the format of a compressed image. A compressed image consists of a header followed by a body. The variable-length header encodes image and compression parameters. The body consists of losslessly encoded mapped prediction residuals, $\delta_{z,y,x}$ from the predictor. The mapped prediction residuals are sequentially encoded in the orderselected by the user and indicated in the header. This encoding order need not correspond to the order in which samples are output from the imaging instrument or processed by the predictor. To encode the mapped prediction residuals for an image, a user may choose to use the sample adaptive entropy coding approach or the block-adaptive approach. The sample-adaptive entropy coder typically yields smaller compressed images than the block adaptive entropy coder.

Under the **sample-adaptive entropy** coding approach, each mapped prediction residual is encoded using a variable-length binary codeword. The variable-length codes used are adaptively selected based on statistics that are updated after each sample is encoded. Separate statistics are maintained for each spectral band, and the compressed image size does not depend on the order in which mapped prediction residuals are encoded.

Under the **block-adaptive entropy** coding approach, the sequence of mapped prediction residuals is partitioned into short blocks, and the encoding method used is independently and adaptively selected for each block. Depending on the encoding order, the mapped prediction residuals in a block may be from the same or different spectral bands, and thus the compressed image size depends on the encoding order when this method is used.

Compressed Image shall consist of a variable-length header followed by variable length body as shown in Fig. 5.1

Fig. 5.1 Compressed Image Structure

5.1 BIT NUMBERING CONVENTION AND NOMENCLATURE

In this section, the following convention is used to identify each bit in an N-bit field. The first bit in the field to be transmitted (i.e., the most left justified when drawing a figure) is defined to be 'Bit 0'; the following bit is defined to be 'Bit 1' and so on up to 'Bit N-1'. When the field is used to express a binary value (such as a counter), the Most Significant Bit (MSB) shall be the first transmitted bit of the field; i.e., 'Bit 0'. This is represented on the Fig. 5.2.

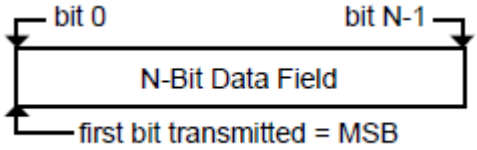


Fig. 5.2 Bit numbering convention

In accordance with modern data communications practice, spacecraft data fields are often grouped into 8-bit 'words' which conform to the above convention. Throughout this project, the following nomenclature is used to describe this grouping:

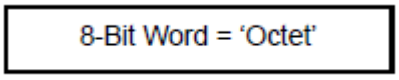


Fig. 5.3 Nomenclature

5.2 OVERVIEW

This chapter defines for standardization a particular adaptive source encoding algorithm that has widespread applicability to many forms of digital data. In particular, the science data from many types of imaging or non-imaging instruments are well suited for the application of this algorithm.

A Lossless source encoding technique preserves source data accuracy and removes redundancy in the data source. In the decoding process, the original data can be reconstructed from the compressed data by restoring the removed redundancy; the decompression process adds no distortion. This technique is particularly useful when data integrity cannot be

compromised. The price it pays is generally a lower Compression Ratio, which is defined as the ratio of the number of original uncompressed bits to the number of compressed bits including overhead bits necessary for signaling parameters.

5.2.1 THE SOURCE CODER

The Lossless source coder consists of two separate functional parts: the preprocessor and the adaptive entropy coder, as shown in Fig. 5.4.

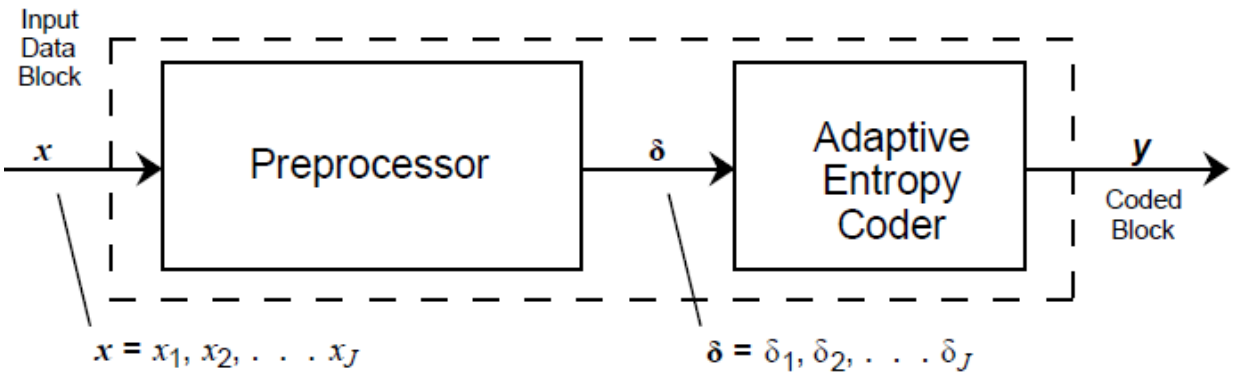


Fig. 5.4 Schematic of the Source Coder

The preprocessor used in our project is Prediction method. The inputs to the source coder are

$$x = x_1, x_2, \ldots, x_J \quad (5.1)$$

which is a block of J n-bit samples, where n is a constant value.

Preprocessor:

The preprocessor applies a reversible function to input data samples $\boldsymbol{x}$, to produce a 'preferred source':

$$\delta = \delta_1, \delta_2, \ldots . \delta_i, \ldots, \delta_J \quad (5.2)$$

where each δ_i is an n-bit integer, $0 \leq \delta_i \leq (2^n - 1)$. For an ideal preprocessing stage, δ will have the following properties:

a) the $\{\delta_i\}$ is statistically independent and identically distributed;

b) the preferred probability, p_m, that any sample δ_i will take on integer value m is anon-increasing function of value m, for $m = 0, 1, \ldots (2^n - 1)$.

The preprocessor function is a reversible operation, and, in general, the best Lossless preprocessor will meet the above conditions and produce the lowest entropy, which is a measure of the smallest average number of bits that can be used to represent each sample.

Adaptive Entropy Coder:

The function of the Adaptive Entropy Coder is to calculate uniquely decipherable, variable length code words corresponding to each block of samples input from the preprocessor. The entropy coder incorporates multiple coding options, each exhibiting efficient performance over different yet overlapping ranges of entropy. The coder selects the coding option that gives the highest compression ratio among the various options on the same block of *J* samples. A code option 'identifier', requiring only a few bits, is attached before the first codeword bit in a coded block to signal the coding option to the decoder for proper decompression. Since the block size *J* is small (16 or fewer samples) and a new code option is selected for each block, the overall coding can be adjusted to adapt to rapid changes in data statistics.

Almost all data compression methods involve the use of a model, a prediction of the composition of the data. When the data matches the prediction made by the model, the encoder can usually transmit the content of the data at a lower information cost, by making reference to the model. This general statement is a bit misleading as general data compression algorithms would include the popular LZW and LZ77 algorithms, which are hardly comparable to compression techniques typically called adaptive. Run length encoding and the typical JPEG compression with run length encoding and predefined Huffman codes do not transmit a model. A lot of other methods adapt their model to the current file and need to transmit it in addition to the encoded data, because both the encoder and the decoder need to use the model.

As the data is transmitted, both encoder and decoder adapt their models, so that unless the character of the data changes radically, the model becomes better adapted to the data it is handling and compresses it more efficiently approaching the efficiency of the static coding.

5.3 ADAPTIVE ENTROPY CODER

5.3.1 CODE SPECIFICATION

Fig. 5.4 represents the general-purpose Adaptive Entropy Coder with a preprocessor. Basically, such a coder chooses one of a set of code options to use to represent an incoming block of preprocessed data samples, δ. A unique identifier (ID) bit sequence is attached to the code block to indicate to the decoder which decoding option to use and shown in Fig. 5.5.

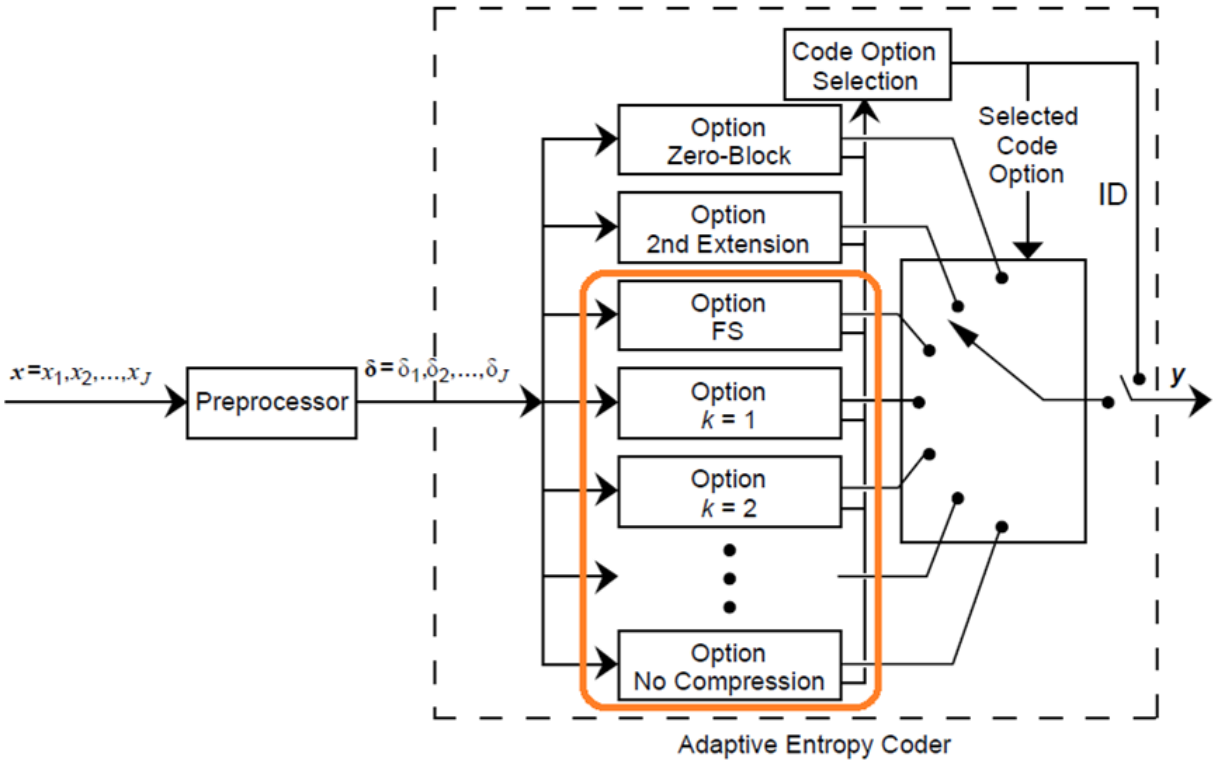


Fig. 5.5 The Adaptive Entropy Coder with a Preprocessor

The basic code selected is a variable-length code that utilizes Rice's adaptive coding technique. In Rice's coding technique, several algorithms are concurrently applied to a block of *J* consecutive preprocessed samples. The algorithm option that yields the shortest codeword sequence for the current block of data is selected for transmission. The zero-block option is a special case in that a single codeword sequence represents one or more consecutive blocks of *J* preprocessed samples. In all other options, the codeword sequence represents a single block of *J* consecutive preprocessed samples.

The following variables are required by Rice's adaptive coding technique:

i. block size, *J*;

ii. resolution, *n* (number of input bits/sample);

iii. the ID bit sequence of the selected code option.

The following constraints shall apply to the Entropy Coder's variable-length adaptive coding scheme:

J = 8 or 16 samples per block;

n = resolution with a maximum of 32 bits per sample with digital signal values from 0 to 2^n-1, or from -2^{n-1} to 2^{n-1}-1.

5.3.2 FUNDAMENTAL SEQUENCE

The most basic option is a variable-length Fundamental Sequence (FS) codeword, which consists of *m* zeros followed by a one when preprocessed sample $\delta_i = m$. Table 5.1 illustrates the FS codewords. A Fundamental Sequence is the concatenation of '*J*' FS codewords.

Table 5.1 Fundamental Sequence codewords as a Function of the Preprocessed Samples

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

5.3.3 SAMPLE SPLITTING

The k^{th} split-sample option is obtained by removing the k least-significant bits (LSBs) from the binary representation of each preprocessed sample, δ_i and encoding the remaining bits with an FS codeword (see Fig. 5.6). This produces a varying codeword length.

The FS codewords for the current block of J preprocessed samples are transmitted along with the removed LSBs, preceded by an ID field indicating the value of k. This process enables the adaptation of codeword length to source-data statistics.

The FS option described in 5.3.2 is a special case of sample splitting where $k = 0$.

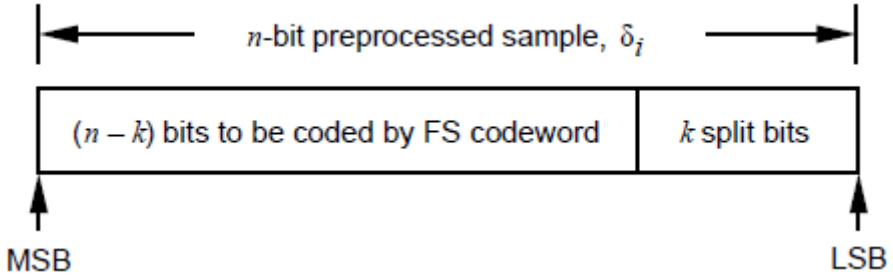


Fig. 5.6 Split-Sample Format

Sample splitting is an important aspect of encoding because it decides how many bits has to be encoded using the FS Codeword and how many bits are stored as residual bits to be transmitted later. Fig. 5.6 gives the sample splitting format for the incoming sample. First the desired k-option is chosen using an *if-else* construct. The *if-else* statement helps us to find the least length codeword for all the samples for every k-option. The k-option corresponding to the least codeword length is the chosen or the desired encoding option. n is the number of bits of the incoming preprocessed sample. $(n - k)$ bits are taken separately and coded by FS codeword. The remaining k bits are kept as it is, and are attached to the $(n - k)$ coded bits to be transmitted together along with the Identification Key (ID) that specifies which coding option has been chosen.

In our project, the value of *k* has been varied from **0** to **14**. The sample splitting is done in VHDL using a **CASE STATEMENT**:-

```
P4: process(k_option,reset,clock,imapped_PR)
    begin
        --if  clock'event and clock = '0' then

            case k_option is
                when 0 => delta(conv_integer(Cnt)) <= (imapped_PR(15 downto 0));
                          LSB(conv_integer(Cnt)) <= (others => '0');

                when 1 => delta(conv_integer(Cnt)) <= ('0' & imapped_PR(15 downto 1));
                          LSB(conv_integer(Cnt)) <= ("0000000000000" & imapped_PR(0));

                when 2 => delta(conv_integer(Cnt)) <= ("00" & imapped_PR(15 downto 2));
                          LSB(conv_integer(Cnt)) <= ("000000000000" & imapped_PR(1 downto 0));


    when 3 => delta(conv_integer(Cnt)) <= ("000" & imapped_PR(15 downto 3));
              LSB(conv_integer(Cnt)) <= ("00000000000" & imapped_PR(2 downto 0));

    when 4 => delta(conv_integer(Cnt)) <= ("0000" & imapped_PR(15 downto 4));
              LSB(conv_integer(Cnt)) <= ("0000000000" & imapped_PR(3 downto 0));

    when 5 => delta(conv_integer(Cnt)) <= ("00000" & imapped_PR(15 downto 5));
              LSB(conv_integer(Cnt)) <= ("000000000" & imapped_PR(4 downto 0));

    when 6 => delta(conv_integer(Cnt)) <= ("000000" & imapped_PR(15 downto 6));
              LSB(conv_integer(Cnt)) <= ("00000000" & imapped_PR(5 downto 0));

    when 7 => delta(conv_integer(Cnt)) <= ("0000000" & imapped_PR(15 downto 7));
              LSB(conv_integer(Cnt)) <= ("0000000" & imapped_PR(6 downto 0));

    when 8 => delta(conv_integer(Cnt)) <= ("00000000" & imapped_PR(15 downto 8));
              LSB(conv_integer(Cnt)) <= ("000000" & imapped_PR(7 downto 0));

    when 9 => delta(conv_integer(Cnt)) <= ("0000000000" & imapped_PR(15 downto 9));
              LSB(conv_integer(Cnt)) <= ("00000" & imapped_PR(8 downto 0));

    when 10 => delta(conv_integer(Cnt)) <= ("0000000000" & imapped_PR(15 downto 10));
               LSB(conv_integer(Cnt)) <= ("0000" & imapped_PR(9 downto 0));

    when 11 => delta(conv_integer(Cnt)) <= ("00000000000" & imapped_PR(15 downto 11));
               LSB(conv_integer(Cnt)) <= ("000" & imapped_PR(10 downto 0));

    when 12 => delta(conv_integer(Cnt)) <= ("000000000000" & imapped_PR(15 downto 12));
               LSB(conv_integer(Cnt)) <= ("00" & imapped_PR(11 downto 0));

    when 13 => delta(conv_integer(Cnt)) <= ("0000000000000" & imapped_PR(15 downto 13));
               LSB(conv_integer(Cnt)) <= ('0' & imapped_PR(12 downto 0));

    when 14 => delta(conv_integer(Cnt)) <= ("00000000000000" & imapped_PR(15 downto 14));
               LSB(conv_integer(Cnt)) <= imapped_PR(13 downto 0);

    when others => delta(conv_integer(Cnt)) <= (others => '0');
                   LSB(conv_integer(Cnt)) <= (others => '0');
end case;
```

where ‘delta’ is the ($n - k$) bits coded with FS and ‘LSB’ is the k split bits.

5.3.4 NO COMPRESSION

The last option is not to apply any data compression. If it is the selected option, the preprocessed block of samples receives an attached identification field but is otherwise unaltered.

5.3.5 CODE SELECTION

The Adaptive Entropy Coder includes a code selection function, which selects the coding option that performs best on the current block of samples. The selection is made on the basis of the number of bits that the selected option will use to code the current block of samples. An ID bit sequence specifies which option was used to encode the accompanying set of codewords. The ID bit sequences are shown in table 5.1.

5.4 DATA FORMAT

5.4.1 LOSSLESS DATA STRUCTURES

Several parameters are required in order to transfer the adaptive variable-length losslessly coded data between the coder and the telemetry channel packet formatter.

OPTION IDENTIFICATION:

The ID Field specifies which of the options was used for the accompanying set of samples. The ID-code keys for each of the options are shown in Table 5.2. It depicts the Identification key (ID) selection procedure for the input number of bits *n*.

For applications not requiring the full entropy range of performance provided by the specified options, a subset of the options at the source may be implemented. The ID is always required, even if a subset of the options is used.

Table 5.2 Selected Code Option Identification Key

Option	$n \leq 8$	$8 < n \leq 16$	$16 < n$
Zero Block	0000	00000	000000
Second Extension	0001	00001	000001
FS	001	0001	00001
$k = 1$	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	111	1111	11111
NOTE – '—' indicates no applicable value.			

REFERENCE SAMPLE

When the preprocessor is present and the reference sample is required, the first CDS of the Source Packet Data Field shall contain a reference sample. References shall then be inserted in the Source Packet Data Field at least every 256 CDSes. When the preprocessor is absent, or it does not require a reference sample, the reference sample shall not be inserted in the CDS.

5.4.2 CODED DATA SET FORMAT

The CDS format when a sample-splitting option is selected is shown in Fig. 5.7. Fig. 5.7 a shows the case where there is a reference sample; Fig. 5.7b shows the format when no reference sample is present. The CDS has the following structure when a sample-splitting option is selected: 1) ID bit sequence optionally followed by an *n*-bit reference sample, 2) compressed data, and 3) concatenated *k* least-significant bits from each sample.

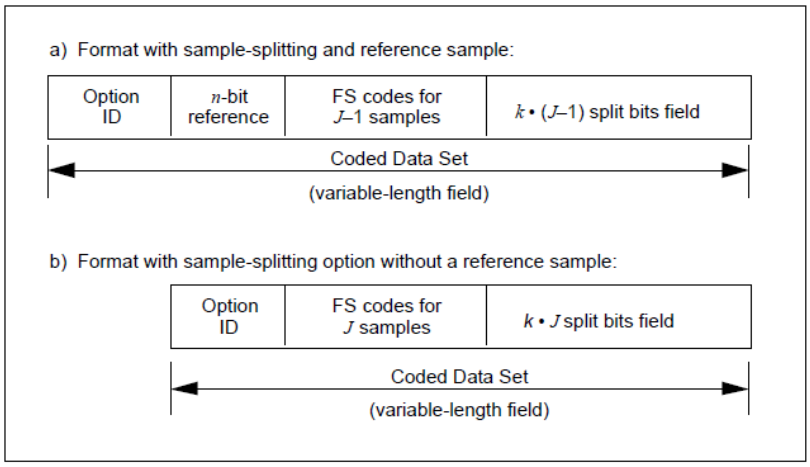


Fig. 5.7 CDS Format When Sample-Splitting Option Is Selected

When the no-compression option is selected, the CDS is fixed length containing the option ID field, optionally followed by an *n*-bit reference sample, and *J* preprocessed samples. The case where a reference is present is shown in Fig. 5.8a; the non-reference case is shown in Fig. 5.8b.

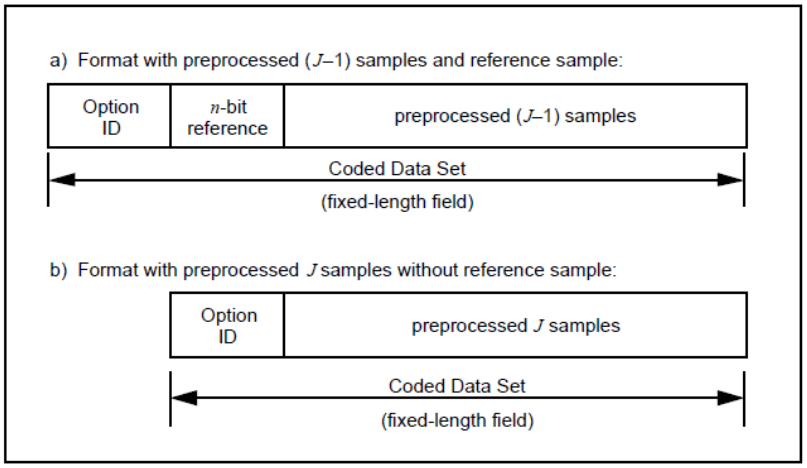


Fig. 5.8 CDS Format When No-Compression Option Is Selected

5.5 FINITE STATE MACHINE (FSM) TO CHECK THE TRANSMISSION

After choosing the desired *k*-option and sample splitting the encoded sample has to be transmitted. The block adaptive encoder is designed in such a way that it allows the transmission of only 16 – bit coded words.

To check the transmission of these codewords a finite state machine (FSM) has been designed which effectively shows the transitions between the states based on the number of bits present in the sample. The state level diagram of the finite state machine is shown in Fig. 5.9

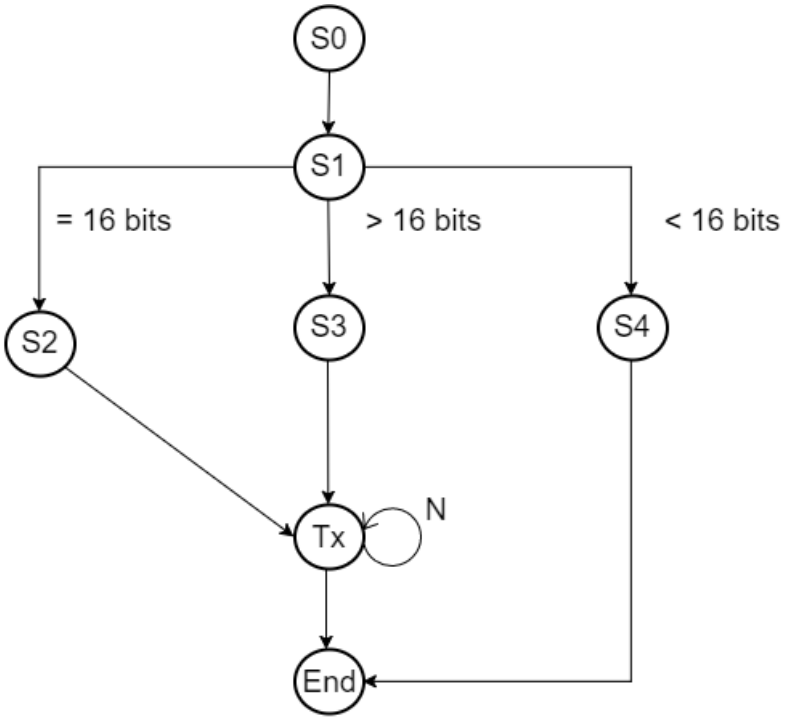


Fig. 5.9 FSM State Diagram

The working of the finite state machine can be explained in steps:

State S0: The input FS codewords are read in this state. S0 is called the IDLE state.

State S1: The number of bits of the input sample are checked. Based on that the next state will be S2 or S3 or S4.

State S2: If the input number of bits is exactly equal to 16 then the next state will be S2. Then the data is transmitted directly.

State S3: If input number of bits is greater than 16 the next state will be S3. The LSB 16 bits are transmitted. And the remaining bits are stored in a residual code word Register (16 bit register).

State S4: If input number of bits is lesser than 16 then the next state is S4. In this state no codeword is transmitted. Instead the entire codeword is stored as a residual in a residual code word register (which is a 16 bit register) to be transmitted later.

The **Total Number of bits** in state S1 is updated by adding the residual bits from state S3 and S4 from previous states. Once the total number of bits becomes 16 it is transmitted.

State Tx: If the **Current_Code_Word** register or **Residual_Code_Word** register has a 16-bit data, then that is sent to Transmission stage and transmitted after waiting for N cycles.

Finally, **End Transmission** state ends the transmission process for that block of 8 samples.

The FSM described above is realised in VHDL using the code given below:

```
-- FSM combinational block
fsm: process (Clock,reset,next_state,Start_FSQ)
        begin
             if reset = '0' then
                next_state <= S0;
             elsif Clock'event and Clock = '1' then
                case current_state is
                    when S0 =>  if(Start_FSQ = '1') then
                                    next_state <= S1;
                                else
                                    next_state <= S0;
                                end if;

                    when S1 =>
                            if(Total_bits = "0000000000010000") then
                                next_state <= S2;
                            elsif(Total_bits > "0000000000010000") then
                                next_state <= S3;
                            elsif(Total_bits < "0000000000010000") then
                                next_state <= S4;
                            end if;


                    when S2 =>  next_state  <= Transmission;

                    when S3 =>  next_state  <= Transmission;

                    when S4 =>  if(Count_1 = "000") then
                                    next_state <= End_Transmission;
                                else
                                    next_state <= S1;
                                end if;

                    when Transmission =>
                                if(N_Count = conv_integer(Number_of_cycles) - 1) then
                                    next_state <= End_Transmission;
                                else
                                    next_state <= Transmission;
                                end if;

                    when End_Transmission =>
                                if(Count_1 = "000" ) then
                                    next_state <= S0;
                                else
                                    next_state <= S1;
                                end if;

                end case;
```

CHAPTER 6

SIMULATION AND RESULTS

CHAPTER 6

SIMULATION AND RESULTS

6.1 CODING ENVIRONMENT

This project aims at software implementation of the **CCSDS 123** Algorithm for Lossless Hyperspectral Compression in **VHDL**, analyzing different algorithm parameters with respect to compression parameters and validation of the data. The entire compression algorithm has been fragmented into multiple modules, with the **predictor** and **encoder** being two main entities of the compression standard. The hyperspectral image is fed as an input to the predictor whose output serves as a driving input to the encoder. The encoder output is a compressed hyperspectral image.

Both the predictor and encoder codes comprise of several software modules in cascade. Finally the entire code has been integrated with the help of a Top Module in VHDL, which receives its input signal drivers from the Top Module Testbench. The clock period assigned to all our inputs and outputs is **15 ns**. The clock frequency is **67 MHz**. The VHDL coding has been done in **Libero Version 9.2** which is a product of **Actel Microsemi®.**

The software implementation results are verified and compared from the results produced in **Microsoft Visual Studio 2015**. Finally the compression parameters are evaluated for different performance evaluation parameters and the variation in the result is recorded and plotted in MATLAB.

```
library IEEE;                                          -- Library Declarations
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
use work.comps_enc.all;                                -- Package Declaration
```

Fig. 6.1 Library & Package Declarations

6.2 PREDICTOR

Predictor has the following modules which are integrated with the help of a Top Module. The Top Module essentially contains every sub-module declared as a component in it, with the main input – output ports declared in the entity. All the sub-modules are port mapped in the Top Module for internal dataflow.

6.2.1 LOCAL SUM

Local Sum is the first module in the Predictor. The local sum has been calculated for four prediction bands in this project. The code implementation for the same is shown in the figure below. All the outputs in this project have been latched and synchronized with a clock event so as to monitor their variations. Given an input sequence that starts after a reset is set high and at the first positive clock edge, the local sum output appears in the next clock cycle.

```
-- localsum.vhd
library IEEE;                                          -- Library Declarations
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

---------------------------------------------------------------------------------
entity localsum is                                     -- Entity Declaration
port (
    -- ports declaration
          Pnx : in std_logic_vector(15 downto 0);
        PnMxP : in std_logic_vector(15 downto 0);
  clock,reset : in std_logic;
          x,y : in INTEGER RANGE 0 TO 3;
     localSum : out std_logic_vector(17 downto 0)
    );
end localsum;                                          -- End Entity

---------------------------------------------------------------------------------
architecture behavioral of localsum is                 -- Architecture Declaration

-- signal, component etc. declarations
signal a     : std_logic_vector (15 downto 0);
signal b     : std_logic_vector (15 downto 0);
signal c     : std_logic_vector (15 downto 0);
signal d     : std_logic_vector (15 downto 0);
signal z1    : std_logic_vector (17 downto 0);
signal y1    : std_logic_vector (17 downto 0);
signal y2    : std_logic_vector (17 downto 0);
signal y3    : std_logic_vector (17 downto 0);
signal y4    : std_logic_vector (17 downto 0);
constant Nx  : integer := 4;
signal sel   : std_logic_vector(0 to 1);
```

Fig. 6.2 Input specification for Local Sum

From Fig. 6.2 it is seen that the entity list consists of the various input and output ports and their declarations preceded by the library declarations. The input ports are as listed above. The output is the Local Sum.

In addition to the input and output ports there are several signals that serve the purpose of an in-out or a buffer port and hold intermediate values.

```
begin
   -- architecture body

   -- Boundary Conditions Specification for Local Sum
    process(x,y)
    begin

        if(y>0) then
            if (x>0) and (x <(Nx-1))
                then sel<="00"; -- from second row excluding last pixel
            elsif(x=0)
                then sel<="10"; -- fireset pixel from second row onwards
            elsif(x = (Nx-1))
                then sel<="11"; -- last pixel from second row onwards
            end if;
        elsif (y=0)
                then if(x>0)
                    then sel<="01"; -- fireset row
                end if;
        end if;
    end process;

-------------------------------------------------------------------------------

   -- Multiplexer output for Local Sum module
     process(sel,y1,y2,y3,y4) is
    begin

        case sel is
            when "00" => localSum <= y1;
            when "01" => localSum <= y2;
            when "10" => localSum <= y3;
            when "11" => localSum <= y4;
            when others => localSum <= "00000000000000000";
        end case;

    end process;
```

Fig. 6.3 Boundary conditions and output selection using a MUX

The local sum for every input sample value is calculated based on certain boundary conditions specifications. We have implemented those conditions using an if-else construct and the desired output for the appropriate boundary condition is derived using a 4:1 MULTIPLEXER as shown in Fig. 6.3.

Every input for calculating the local sum is actually the neighboring pixel of the pixel under consideration. The prediction algorithm itself is based on the neighboring pixel values. Hence a sensible method to choose the neighboring pixels for every changing input pixel is assigning a clock delay to input pixel's position. The four pixels taken into account are the West, North, North-west and the North-East pixel. Hence the Fig. 6.4 and 6.5 below show the generation of the neighboring pixel positions using a clock delay.

```
--- DFF to get one clock delay (rising to rising) to generate inputs a,b,c,d
P0: process(clock,reset,Pnx) begin                          -- Pnx given to input 'a'
        if (reset='0') then
                a <= "0000000000000000";
        elsif (clock'event and clock = '1')
                then a <= Pnx;
        end if;
    end process;

P1: process(clock,reset,PnMxP) begin                        -- PnMxP given to input 'd'
        d <= PnMxP;
        if (reset='0') then
            c <= "0000000000000000";
        elsif (clock'event and clock = '1')
            then c <= d;                                    -- 'd' given to 'c' with one cycle delay
                 b <= c;                                    -- 'c' given to 'b' with one cycle delay
        end if;
    end process;

-------------------------------------------------------------------------------------------------
-- Output Conditions for Local Sum : MUX Output
        y1 <= ("00" & a) + ("00" & b) + ("00" & c) + ("00" & d);
        y2 <= ((a(15 downto 0) & "00"));
        z1 <= ("00" & c) + ("00" & d);
        y3 <= (z1(16 downto 0) & '0');
        y4 <= ("00" & a) + ("00" & b) + ("00" & c) + ("00" & c);
```

Fig. 6.4 Assigning clock delay to neighboring inputs& MUX output

Finally the output is verified using the results obtained in Microsoft Visual Studio 2015.

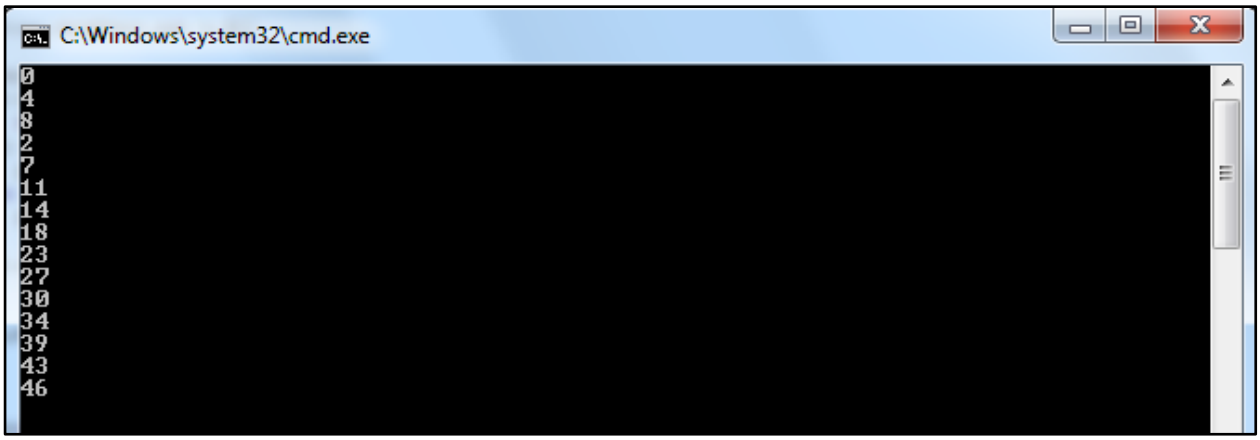


Fig. 6.5 Output Verification of Local Sum

6.2.2 LOCAL DIFFERENCE AND DIFFERENCE VECTOR

The output of the local sum module drives the input for the local difference module. There are 2 types of local differences namely, Central Local Difference and Directional Local Differences. The formula for these parameters have been mentioned in the previous sections and they are realized using VHDL in the following way as shown in the figure below.

```
    -- Calculation of Central Local Difference
process(x,y,localSum)
begin
    if (x = 0) and (y = 0)
        then central_LD <= "00000000000000000";
    else
        central_LD <= ('0' & ((Pnx(15 downto 0)) & "00")) - ('0' & localSum);
    end if;
end process;

    -- Calculation of Directional Local Differences
process(x,y,localSum)
begin
    if (x = 0) and (y > 0) then
        dir_N <=('0' & (c(15 downto 0) & "00")) - ('0' & localSum);
    elsif (x > 0) and (y > 0) then
        dir_N <= ('0' & (c(15 downto 0) & "00")) - ('0' & localSum);
    else
        dir_N <= "00000000000000000";
    end if;
end process;

process(x,y,localSum)
begin
    if (x > 0) and (y > 0) then
            dir_W  <= ('0' & (a(15 downto 0) & "00")) - ('0' & localSum);
            dir_NW <= ('0' & (b(15 downto 0) & "00")) - ('0' & localSum);
    elsif(x = 0) and (y > 0) then
            dir_W  <= ('0' & (c(15 downto 0) & "00")) - ('0' & localSum);
            dir_NW <= ('0' & (c(15 downto 0) & "00")) - ('0' & localSum);
    elsif(y = 0) then
            dir_W  <= "00000000000000000";
            dir_NW <= "00000000000000000";

    end if;
end process;
```

Fig. 6.6 Central and Directional Local Differences

Fig. 6.6 shows that once the local differences are calculated they are stored in a memory which is declared as RAM. Then they are read one by one from the memory to be stored in a vertical vector which is called the Local Difference vector. The local Difference Vector is a very essential component to calculate the predicted central local difference.

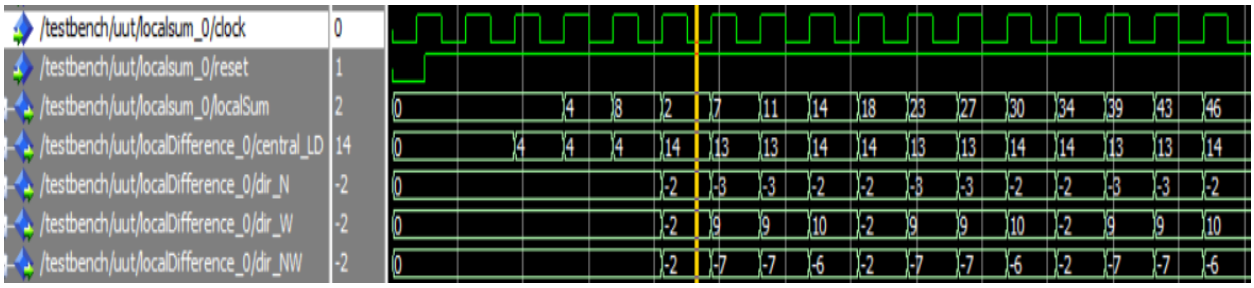


Fig. 6.7 Local sum and Local Difference Simulation for Band 1

6.2.3 WEIGHT INITIALIZATION AND WEIGHT UPDATION

The weight initialization and weight updation operates in a feedback mechanism, wherein the weight values have to be updated for every clock cycle and every input sample in a manner that reduces the Prediction Error. The weight vector consisting of the weight values has to be initialized with the same values in the beginning of every prediction band. The formula for initializing the weights is mentioned the previous section. The simulation waveform shows the weight initialization and updation pattern as shown in Fig. 6.8 and 6.9.

```
P1: process(clock,reset)
    begin
        if reset = '0' then
            Wz_tPlusOne(0) <= "00011100000000000000";-- 458752
            Wz_tPlusOne(1) <= "00000011100000000000";-- 57344
            Wz_tPlusOne(2) <= "00000000011100000000";-- 7168
            Wz_tPlusOne(3) <= "00000000000011100000";-- 896
            Wz_tPlusOne(4) <= (others => '0');        -- 0
            Wz_tPlusOne(5) <= (others => '0');        -- 0
            Wz_tPlusOne(6) <= (others => '0');        -- 0
            weight_signal_L<= '0';

        elsif clock'event and clock = '1' then

            weight_signal_L <= weight_signal;

         if (weight_signal_L = '1') then
            Wz_tPlusOne(0) <= "00011100000000000000";-- 458752
            Wz_tPlusOne(1) <= "00000011100000000000";-- 57344
            Wz_tPlusOne(2) <= "00000000011100000000";-- 7168
            Wz_tPlusOne(3) <= "00000000000011100000";-- 896
            Wz_tPlusOne(4) <= (others => '0');        -- 0
            Wz_tPlusOne(5) <= (others => '0');        -- 0
            Wz_tPlusOne(6) <= (others => '0');        -- 0

        else
            Wz_tPlusOne <= iWz_tPlusOne;
        end if;
        end if;
    end process;
```

Fig. 6.8 Weight initialization and updation

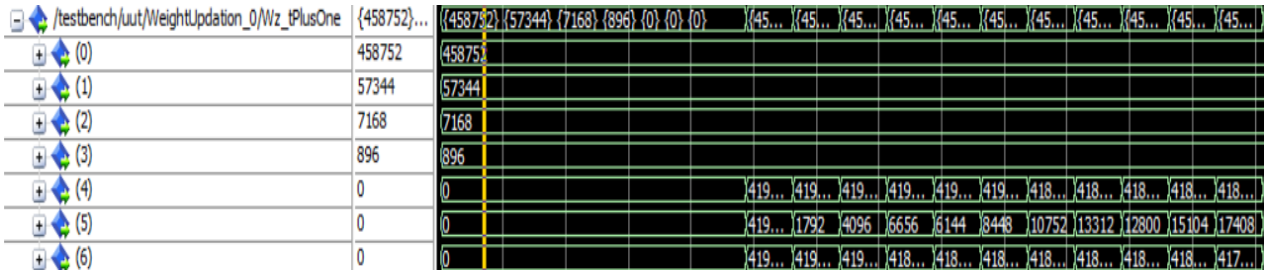


Fig. 6.9 Weight initialization and updation simulation verified

6.2.4 SCALED PREDICTION

Scaled prediction is the next module wherein a CLIP function is used to calculate the Scaled Predicted sample value. The input drivers for this module are: local sum, input sample value, minimum and maximum input sample value, and the difference vector.

Once the scaled predicted value is obtained it is halved using the FLOORING FUNCTION to obtain the Predicted Sample Value.

```
-- Calculation of Scaled Predicted Sample Value for 1st boundary condition
    Sigma <= ('0' & localSum) - ('0' & (SMid(15 downto 0) & "00"));   -- 19 bits

    temp1 <= (Sigma & "00000000000000000");                           -- 38 bits

    temp2 <= temp1(37) & temp1(37) & temp1(37) & temp1(37) &          -- 48 bits
             temp1(37) & temp1(37) & temp1(37) & temp1(37) &
             temp1(37) & temp1(37) & temp1;

    temp3 <= (predicted_CLD + temp2);                                 -- 48 bits

    temp4 <= (temp3(47 downto 20)) + ("00000000000" & (SMid(15 downto 0) & '0'))
             + "00000000000000000000000000001";

-- Boundary Conditions for Scaled Predicted Value
P8: process(reset,clock,pixel_1,pixel_BSQ,Sz_Minus1,Clip_scaled_predicted_value)
    begin
        if reset = '0' then
            scaled_predicted_value <= (others => '0');
        elsif (clock'event and clock = '0')
        then
            if(pixel_BSQ = '1') then
                scaled_predicted_value <= (Sz_Minus1 &'0');
            elsif(pixel_1 = '1') then
                scaled_predicted_value <= (SMid & '0');
            else
                scaled_predicted_value <= Clip_scaled_predicted_value;
            end if;
        end if;
    end process;
```

Fig. 6.10 Scaled Predicted Sample

The calculation of Scaled predicted sample is as depicted in Fig. 6.10, and its simulation is shown in Fig. 6.11.

Fig. 6.11 Scaled Predicted Sample Simulation

6.2.5 MAPPED PREDICTION RESIDUAL

This is the final module in the Prediction Stage. It is also often referred to as the 3rd Level of Prediction. Two parameters called **THETA** and **DELTA** are calculated.

These two parameters are used as conditions for obtaining the output of the prediction stage, called the Mapped Prediction Residual which is as shown below in Fig. 6.12. The mapped prediction residual is a 16-bit unsigned integer which is fed to the Block Adaptive Encoder.

```
P3: process(mod_delta,theta_z,temp_a)
    begin
        if(mod_delta > theta_z) then

            mapped_PR_temp <= ('0' & temp_a);

        elsif(delta_z = 0 or (scaled_predicted_value(0)= '0' and delta_z(16) = '0') or
              (scaled_predicted_value(0)= '1' and delta_z(16) = '1'))then

            mapped_PR_temp <= mod_delta(15 downto 0) & '0';

        else

            mapped_PR_temp <= ((mod_delta(15 downto 0) & '0') - "0000000000000001");

        end if;
    end process;

mapped_PR <= mapped_PR_temp(15 downto 0);
```

Fig. 6.12 Mapped Prediction Residual Code

The simulation of the mapped prediction residual is as shown in the Fig. 6.13 and its software verification is shown in Fig. 6.14.

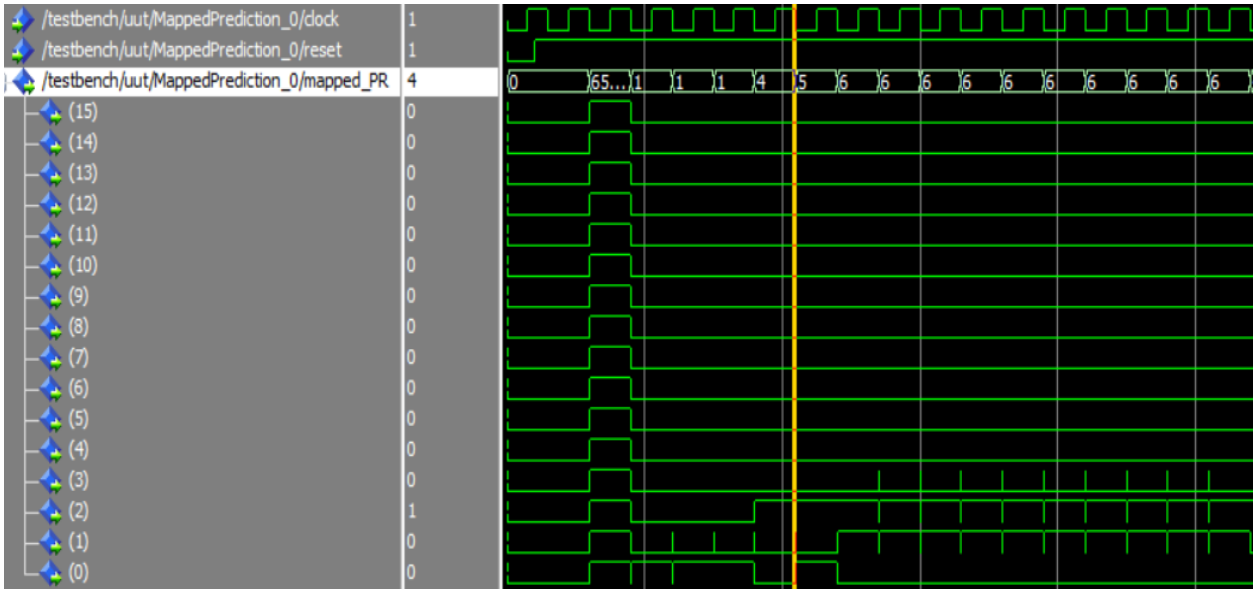


Fig. 6.13 Mapped Prediction Residual Simulation – Final Predictor Output

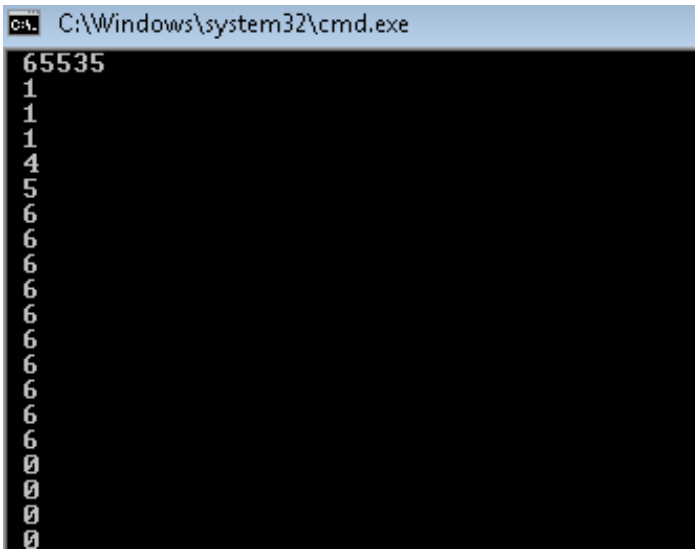


Fig. 6.14 Mapped Prediction Residual Verification

6.3 ENCODER

Encoder has the following modules which are integrated with the help of a Top Module. The Top Module essentially contains every sub-module declared as a component in it, with the main input – output ports declared in the entity. All the sub-modules are port mapped in the Top Module for internal dataflow.

The finite state machine as explained in Section 5.5 has various states. In each state there are certain parameters that are calculated such as: Total number of input bits, Residual bits, Number of cycles to be waited before transmission, etc. These are realized in VHDL as shown in Fig. 6.15.

```
P1: process(reset,next_state,delta,Count_1)
    begin

            if reset = '0' then
                Number_of_cycles  <= (others => '0');
                Residual_Numbits  <= (others => '0');
            else
                case next_state is
                        when S1 =>  Total_bits        <= delta(conv_integer(Count_1)) +
                                    "0000000000000001" + Residual_Numbits;

                        when S2 =>  Number_of_cycles  <= ("00" & Total_bits(15 downto 4));
                                    Residual_Numbits  <= Total_bits(3 downto 0);

                        when S3 =>  Number_of_cycles  <= ("00" & Total_bits(15 downto 4));
                                    Residual_Numbits  <= Total_bits(3 downto 0);

                        when S4 =>  Number_of_cycles  <= ("00" & Total_bits(15 downto 4));
                                    Residual_Numbits  <= Total_bits(3 downto 0);

                    when others =>  Number_of_cycles  <= Number_of_cycles;
                                    Residual_Numbits  <= Residual_Numbits;
                end case;
            end if;

    end process;

P2: process(clock,reset)
    begin
        if(reset = '0') then
            Residual_Numbits_previous <= "0000";
        elsif(clock'event and clock = '1') then
            if(current_state = S2 or current_state = S3 or current_state = S4) then
                Residual_Numbits_previous <= Residual_Numbits;
            end if;
        end if;
    end process;
```

Fig. 6.15 Calculation of Residual bits and Total bits

This simulation waveform shown in Fig. 6.16 is the final output of the block adaptive encoder used.

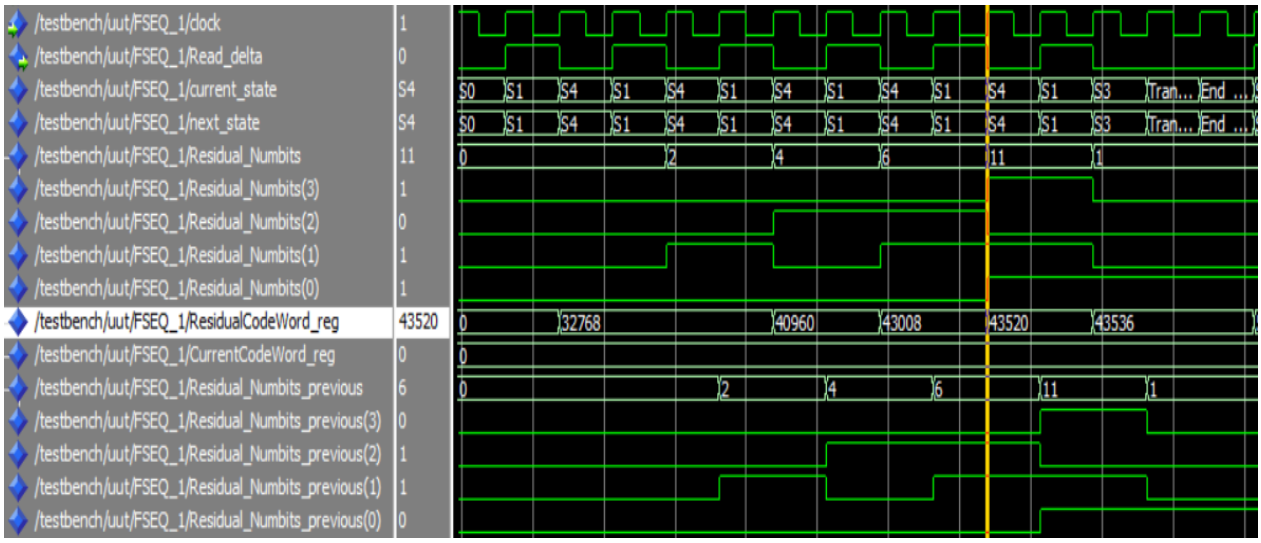


Fig. 6.16 Encoder Output Simulation

CONCLUSION

The presented work shows the implementation of CCSDS 123.0-B-1 standard for the compression of hyperspectral data. The method used for compression is lossless. The implementation is runtime configurable in terms of prediction mode (full), weight update parameters, weight initialization values and image size.

The design and implementation has been configured in the **ProASIC3E** family of a Field Programmable Gate Array with a Die **A3PET600**. The package used is the 484 FBGA. The waveform were simulated in **MODELSIM**.

The compression algorithm designed in this project has been implemented for 4 rows, 4 columns and 4 prediction bands. Hence the magnitude of compression observed is optimum. The outputs have been verified and data has been validated at every intermediate stage and after every individual module. The code written and the simulation output for every module in Predictor and Encoder have been presented in Chapter 6. Also, all the encoded bits are observed to be transmitted as 16-bit residual bits in the residual code word register of the block adaptive encoder. The state transitions during bit-transmission is seen clearly in the simulation waveform of the block adaptive encoder as shown in Fig. 6.15.

It is observed that the compression ratio obtained in Visual C is in the order of **6.0 – 9.3**, provided that the rest of the affecting parameters (Number of Prediction bands, $v_{\min}$, $v_{\max}$, t_{inc})are kept constant. These parameters directly affect the compression rate. The maximum possible compression rate that could be achieved in Visual C, as simulated in Microsoft Visual Studio 2015, was **9.303434** bits/sample. For this case the Prediction duration was observed to be 47.860000 seconds and Encoding duration 22.594000 seconds, making the Overall Compression duration 70.454000 seconds.

FUTURE WORK

Hyperspectral Images are widespread in Remote Sensing and Space Imaging Applications: they are typically three dimensional arrays of images and their raw content may include thousands of bands for a total of over a billion pixels. For adequate storing and transmission over satellite communication links a proper compression scheme is required which has been implemented in software in this project.

The VHDL code in this project implements the compression algorithm only on a few bands (4 here). As a future scope, we plan to extend this algorithm to a large number of bands (e.g. 224) by feeding a hyperspectral image. We plan on using *fread* and *fwrite* functions in C to feed a hyperspectral image.

The future work will also focus on implementing this algorithm on a hardware toolkit, preferably Xilinx FPGA Virtex5QV FX130T to facilitate efficient on-board processing.

BIBILOGRAPHY

[1] *'Lossless hyperspectral and multispectral image Compression'* CCSDS Blue Book.

[2] Giorgio Lopez, Ettore Napoli, Antoni G.M. Strollo, University of Napoli, "FPGA Implementation of the CCSDS-123 B-1 Lossless Hyperspectral Image Compression Algorithm Prediction Stage", *IEEE 6th Latin American Symposium on Circuits & Systems (LASCAS)*, Naples, Italy (2014).

[3] "Lossless data compression," CCSDS 120.0-G-3, pp. 271-350, (April 2013) .

[4] Lucana S., Luis B., Javier M., José F.López, and Roberto S., "Multispectral and Hyperspectral Lossless Compressor for Space Applications (HyLoC): A Low-Complexity FPGA Implementation of the CCSDS 123 Standard", *IEEE Journal Of Selected Topics In Applied Earth Observations And Remote Sensing*, (Feb. 2016).

[5] Jose Enrique Sanchez, Estanislau Auge, Josep Santalo, Ian Blanes and Joan Serra, "Review and implementation of the emerging CCSDS recommended standard for multispectral and hyperspectral lossless image coding," pp.222-228 (2011) .

[6] CHU Qing-Wei et. al., "Research and Software Implementation of CCSDS Lossless Data Compression Algorithm", ICSP2012 Proceedings, Center for Earth Observation and Digital Earth, Chinese Academy of Sciences, Beijing, China(2012).

[7] Dharam Shah, Kuhelika Bera, and Sanjay Joshi, "Software implementation of CCSDS recommended hyperspectral lossless image compression," *I.J. Image, Graphics and Signal Processing*, vol. 4, pp. 35-41 (March 2015).

[8] G. S. M. Weinberger and G. Sapiro "The LOCO-I lossless image compression algorithm: Principles and standardization into JPEG-LS," IEEE Transactions on Image Processing. vol. 9, pp. 1309-1324 (March 2000).