HIGH-PERFORMANCE LOW-COMPLEXITY NEAR-LOSSLESS EMBEDDED MEMORY COMPRESSION FOR HDTV

A Thesis

by

Okan Palaz

Submitted to the
Graduate School of Sciences and Engineering
In Partial Fulfillment of the Requirements for
the Degree of

Master of Science

in the
Department of Computer Science

Özyeğin University
August 2014

HIGH-PERFORMANCE LOW-COMPLEXITY NEAR-LOSSLESS EMBEDDED MEMORY COMPRESSION FOR HDTV

Approved by:

Assoc. Prof. H. Fatih Uğurdağ,
Advisor
Department of Electrical and
Electronics Engineering
Özyeğin University

Asst. Prof. Hasan Sözer
Department of Computer Science
Özyeğin University

Prof. A. Tanju Erdem
Department of Computer Science
Özyeğin University

Dr. Eren Soyak

AirTies Wireless Networks

Date Approved: 6 August 2014

Assoc. Prof. Sezer Gören Uğurdağ
Department of Computer Engineering
Yeditepe University

To my family

ABSTRACT

HDTV video processors need to keep one or more previously scanned frames as they process streaming video when performing tasks such as frame rate conversion, deinterlacing, and other video enhancement techniques. Reading and writing frames require high bandwidth at HD resolutions. This bandwidth can be reduced by applying compression. Video compression methods do not address this problem as they reduce network traffic while adding extra memory traffic. What is needed is high-performance, low-complexity, and lossless (or near-lossless) image compression. This type of compression method is called Embedded Compression (EC). We propose a novel end-to-end embedded memory compression solution. It can support 4K Ultra HD video streams at 30 Hz with a single core implemented in 180nm ASIC technology, which amounts to a per-core pixel rate twice the competition.

ÖZETÇE

HDTV video işlemcileri çerçeve hızı değiştirme, binişme giderme ve diğer video iyileştirme işlemlerini yapabilmek için, bir ya da daha fazla geçmiş çerçeveyi hafızada tutmak durumundadır. HD çözünürlüklerde sıkıştırılmamış çerçeve okuyup yazma işlemi yüksek bant genişliği gerektirir. Bu yüksek bant genişliği, çerçeveleri sıkıştırarak azaltılabilir. Video sıkıştırma yöntemleri bu kullanım için uygun değildir, çünkü bu yöntemler ağ trafiğini azaltırken fazladan hafıza trafiği yaratırlar. Bu amaca yönelik sıkıştırma yönteminin yüksek performanslı, düşük karmaşıklıklı ve kayıpsız (ya da kayıpsıza yakın) olması gerekir. Bu tip sıkıştırma yöntemlerine Gömülü Sıkıştırma (GS) denir. Bu çalışmada, özgün uçtan uca bir gömülü sıkıştırma yöntemi önerilmektedir. Önerilen yöntem, 180nm ASIC üzerinde 4K Ultra HD çözünürlükte 30 Hz frekansında çalışmaya yeterli performansa sahiptir ve çekirdek başına benzer çalışmaların 2 katına yakın saat frekansına erişebilmektedir.

ACKNOWLEDGEMENTS

This thesis was made possible with the help and support of several people.

First, I would like to thank Assoc. Prof. H. Fatih Uğurdağ for his mentorship, guidance, and also his patience throughout both my undergraduate and graduate education years. Thanks to his wisdom that I was able to complete this thesis. He is the one that introduced me to this field and as well as to this project.

I would like to thank Buğra Kertmen, Özgür Özkurt, İsmail Gerçek, and Faruk Dönmez for their time and contributions to this thesis. Their expertise on the subject greatly aided my progress.

I am grateful to Assoc. Prof. Sezer Gören Uğurdağ for her support and for letting me work with her during my undergraduate studies, and also Prof. A. Tanju Erdem and Asst. Prof. Hasan Sözer for taking their time to be on my thesis committee.

I would also like to thank Dr. Eren Soyak for his help on countless various things related to and not related to this thesis.

I am indebted to my colleagues Fatih Temizkan, Gokhan Güner, Anıl Bayram, Furkan Şahin, Mert Kaya, Bilgiday Yüce, and S. Erdem Ulusoy for their friendship.

I would also like to thank The Scientific and Technological Research Council of Turkey (TÜBİTAK) for supporting my studies by offering a scholarship for 2 years.

Finally, I would like to thank my family for their endless support.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

CHAPTER I

INTRODUCTION

This chapter introduces the problem, explains the need for bandwidth reduction in HDTVs, and explains EC method basics.

1.1 Problem Definition

The resolution of TV sets has always had an upward trend. Currently, the most common TV screen resolution is 1920x1080, and the next generation of high resolution TVs, 4K, is on its way to become the next norm in consumer market. A great amount of research is being conducted towards delivering higher resolution content to the end user while keeping the transmission bandwidths low as the delivery channels are still a scarce resource. The upcoming generation of ITU-T video compression standard, HEVC, is able to compress video to half of the size of its predecessor, H.264, at the same visual quality. HEVC is on its way to become the most widely used compression standard in video delivery for both Over-the-Top (OTT) and Digital Video Broadcast (DVB) scenarios. Currently, increasing the efficiency of HEVC encoders is a topic being researched by many.

There is a lot of focus on improving video compression methods to meet the next generation of higher resolution screens. However, another limiting factor regarding the higher resolution is the video processors on the said TVs. Features consumers expect to be implemented in today's HDTVs, such as frame rate conversion, require access to one or more frames that are stored in on-chip or off-chip DRAM. If on-chip DRAM is used, the bottleneck is memory size; while the bottleneck is memory bandwidth when off-chip memory is used. A Full-HD raw frame is about 48Mb and even a 512Mb DRAM chip (the cheapest by today's standards) can hold several of these frames. However, the available memory bandwidth can easily become a major constraint if frames are written and read multiple times. The input memory bandwidth of 1920x1080 pixels per frame at 24 bits per pixel with 60 Hz refresh rate is almost 3 Gbps. To meet the high memory bandwidth requirements, the DRAM chip is often driven by a higher speed clock, which increases power consumption as well as heat generation and makes passing EMC tests harder for consumer electronics products. In some cases, to provide the necessary memory bandwidth a second DRAM is added to the board, which in turn adds to the cost.

1.2 Embedded Compression and Application Scenarios

It is possible to use on-the-fly low complexity image compression methods where raw pixels are processed. Such compression methods that compress the images on-the-fly to reduce the memory bandwidth are called EC methods. EC methods compress the pixel data before they are written to DRAM. These methods are often used in video and image compressor codec ICs for access to reconstructed frame pixel data in the compression loop for search operations and the like.

It is worth noting that video compression methods (e.g., MPEG, H.264) do not address this problem as they reduce network traffic while adding extra memory traffic. Traditional image compression methods like JPEG and PNG are not applicable to this scenario either, as they aim for hard-disk or flash memory storage reduction and they also introduce extra memory traffic.

Some properties of JPEG-LS [1] are suitable as it has a low memory size requirement and is low complexity. However, the context modeling of JPEG-LS creates a data dependency between pixels, which makes it unsuitable for high speed real-time applications. It also lacks a rate control system.

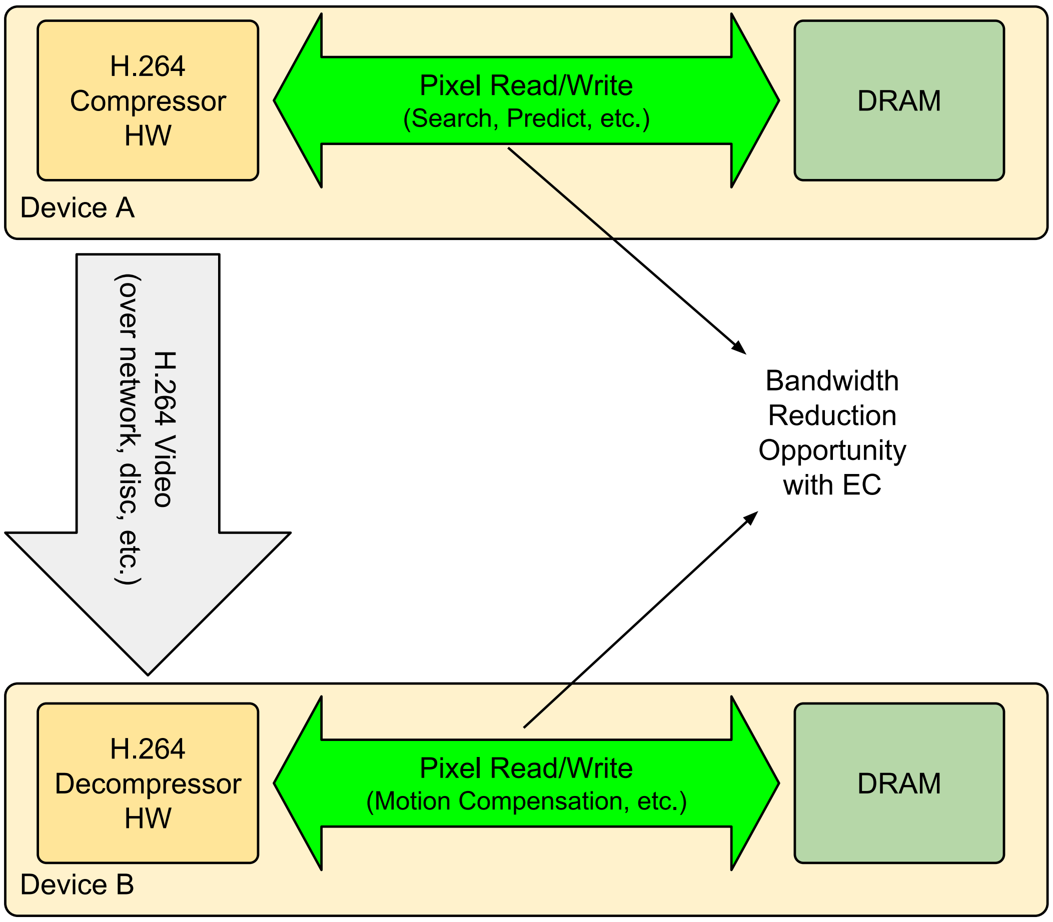
H.264 Compressor HW
Pixel Read/Write (Search, Predict, etc.)
DRAM
Device A
H.264 Video (over network, disc, etc.)
Bandwidth Reduction Opportunity with EC
H.264 Decompressor HW
Pixel Read/Write (Motion Compensation, etc.)
DRAM
Device B

Figure 1: Using EC in Video Compressor ICs.

1.2.1 The Use of Embedded Compression in Video Compressor ICs

Video compression methods such as MPEG and H.264 compress frame data in pixel blocks. For inter-predicted blocks a search is carried out using one or more frames. A good encoder finds a relatively good match for the block. Depending on the efficiency of the encoder this operation may become shorter.

Software based encoders are used for scenarios such as compressing movies for Blu-Ray discs where there is little to no time constraint and image quality to bitrate ratio is critical. Since timing is not critical, the search and rate control operations may even be done exhaustively to get the best possible image quality to bitrate ratio.

Hardware based compressors are often used for other scenarios such as live streaming, recording or transcoding where timing is critical. In such cases, compression is not ideal; the primary objective is meeting the timing requirement. Compression quality, while also important, is secondary.

It is worth pointing out the difference between what the video compression and EC are targeting. The diagram in Fig. 1 aims to show the difference between the objectives. The objective of the H.264 encoder is to compress the video stream so that the network transfer bandwidth or the final storage size is lower. To create this compressed video file, the raw video is processed heavily. The heavy processing creates a lot of bandwidth, because from a bandwidth perspective block search operation in video compression methods is very costly as it requires access to multiple frames in a random manner. Some sacrifices may be made to reduce the number of read operations. These include limiting the search area or reducing the time spent searching, at the cost of degrading the compression quality. Another way to reduce memory bandwidth is compressing the pixel data with EC methods before writing to memory and decompressing before reading the blocks. It should be noted that if the EC method used here is lossy, it will add to the total distortion of the final compressed video.

1.2.2 Using Embedded Compression in HDTV ICs

Similar to the compressor ICs, HDTVs also deal with uncompressed pixel data at high resolutions. HDTVs often store incoming frames for post-processing purposes. Two of the most commonly done operations are deinterlacing and frame rate conversion.

1. Deinterlacing: HDTV screens are updated in a progressive manner, so interlaced video content needs to be converted into progressive in order to be displayed. As such, deinterlacing is one of the most common operations performed in HDTV video processor ICs as many devices still output interlaced video. The most common sources of interlaced video are analog TV broadcast and legacy devices that output 1080i video. Deinterlacing operation requires access to multiple frames.

2. Frame Rate Conversion: TV screens with fixed refresh rates need to convert frame rates to their native refresh rates to properly display the video. The trivial scenario is when the native refresh rate is a multiple of the input frame rate, such as the common scenario of displaying 30 Hz video on a 60 Hz screen. However, there are cases when the native refresh rate is not a multiple of the input frame rate. This occurs commonly with movies, as they are mostly filmed at 24 Hz. The most widely used solution to this problem is displaying frames on screen for unequal time periods.

Being as such, in a common HDTV scenario, multiple modules may need access to frame data on the video processor IC similar to the scenario shown in Fig. 2. This creates a lot of bandwidth as the input to the IC comes from a continuous stream of high resolution frames usually at 60 Hz. For the most common resolution today, 1080p at 24 bits per pixel, the input bandwidth is at around 3 Gbps per second and for 4K Ultra HD resolution at the same refresh rate the bandwidth is at around 12

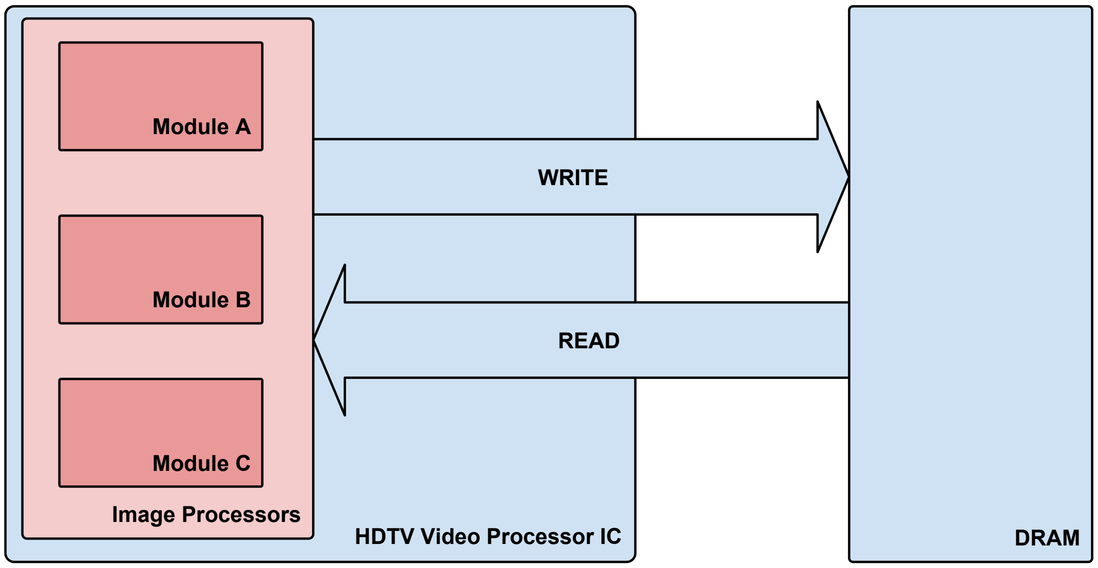

Figure 2: A typical scenario when modules access frame data.

Gbps. Similar to the compression codec scenario, there is an opportunity to decrease this bandwidth by applying EC as shown in Fig. 3.

1.3 Previous Work

There exists pipelined JPEG-LS based encoders such as [2], [3], and [4]. However, they do not propose a decoder implementation and they lack a rate control system. EC methods proposed in [5], [6], [7], and [8] are designed to be used in lossy video scenarios, where they compress the reconstructed lossy pixels in video or image codecs such as MPEG-2, H.264, and JPEG2000. Consequently, they are not applicable to the raster scan input scenario on HDTVs. [9] proposes an EC method suitable for raster scan input. Their method requires multiple compressor and decompressor cores to meet the throughput requirements.

1.4 Characteristics of the Proposed Method

We propose a small footprint compression method for raster scan input/output scenarios for compressing the memory bandwidth to fixed Target Compression Ratio

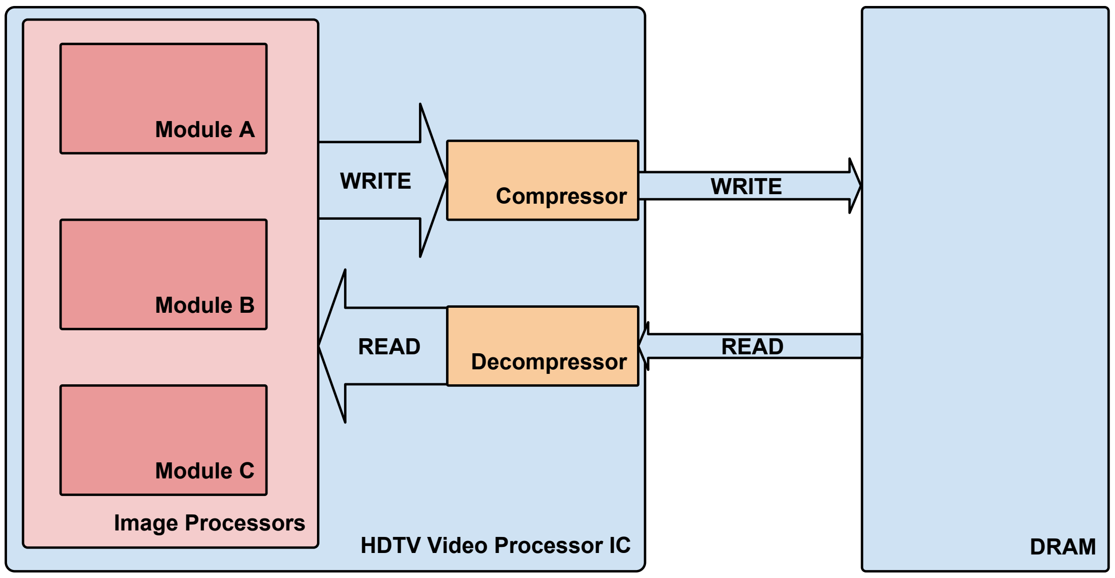

Figure 3: Using EC in HDTV ICs.

(TCR) values. One of the main objectives of the proposed design is to be able to meet the throughput requirements for processing 1920x1080 resolution frames at 60 Hz refresh rate on a Xilinx Spartan-6 FPGA. The proposed method has the following characteristics.

1. The timing requirements are strict. The chip is driven by a single clock and the clock speeds are defined by the resolution standards.

2. The compression needs to be done on raster scan input. The pipeline cannot be stalled and there cannot be multiple passes.

3. The compression needs to be failsafe as failing would ultimately mean corrupting the output on the TV screen and is unacceptable.

4. The bandwidth needs to be compressed to an absolute TCR regardless of the video content being compressed.

5. The compression needs to be at near-lossless levels of visual quality.

6. The implementation of the algorithm needs to meet the timing requirements for being able to compress 1920x1080 input at 60 Hz refresh rate on the target FPGA, which is low cost Xilinx Spartan-6 chip.

1.5 Outline

This thesis is organized as follows. Chapter 2 describes the proposed algorithm and the proposed design in detail. Chapter 3 presents performance evaluation tests and synthesis results and discusses the results. Chapter 4 discusses possible improvements to the proposed method. Finally, Chapter 5 gives a brief conclusion.

CHAPTER II

PROPOSED METHOD

In this Chapter, the algorithm and the proposed design are described in detail. In section 2.1, compressor and decompressor modules and their submodules are briefly explained. Sections 2.2 and 2.3 describe the quantization and prediction methods, respectively. The entropy coding method is explained in section 2.4. Rate control method used to prevent buffer overflow and underflows are described in section 2.5. Multiple stream generation technique used to increase the performance is detailed in section 2.7, and bitstream multiplexing method is described in section 2.8.

2.1 Design Overview

At the top level of the design there are 2 submodules: compressor and decompressor. The following sections go into detail on how each of them is structured.

2.1.1 Compressor Architecture

Compressor consists of the modules shown in Fig. 4. The compression is based on prediction and residual coding. For each pixel a prediction is made and the residual is encoded with a variant of Golomb-Rice coding. The variable length Golomb-Rice symbols are concatenated in the bitstream generator module and divided into chunks of 32-bit words. The 32-bit words are pushed into the bitstream buffer, which is depleted at the target bitrate. Rate control module is responsible for the decision of going into the lossy mode for preventing overflows in the bitstream buffer. For prediction, the previously scanned line is buffered.

As there are three color channels and they are compressed independently, three instances of the compressor are needed. This results in three separate bitstreams.

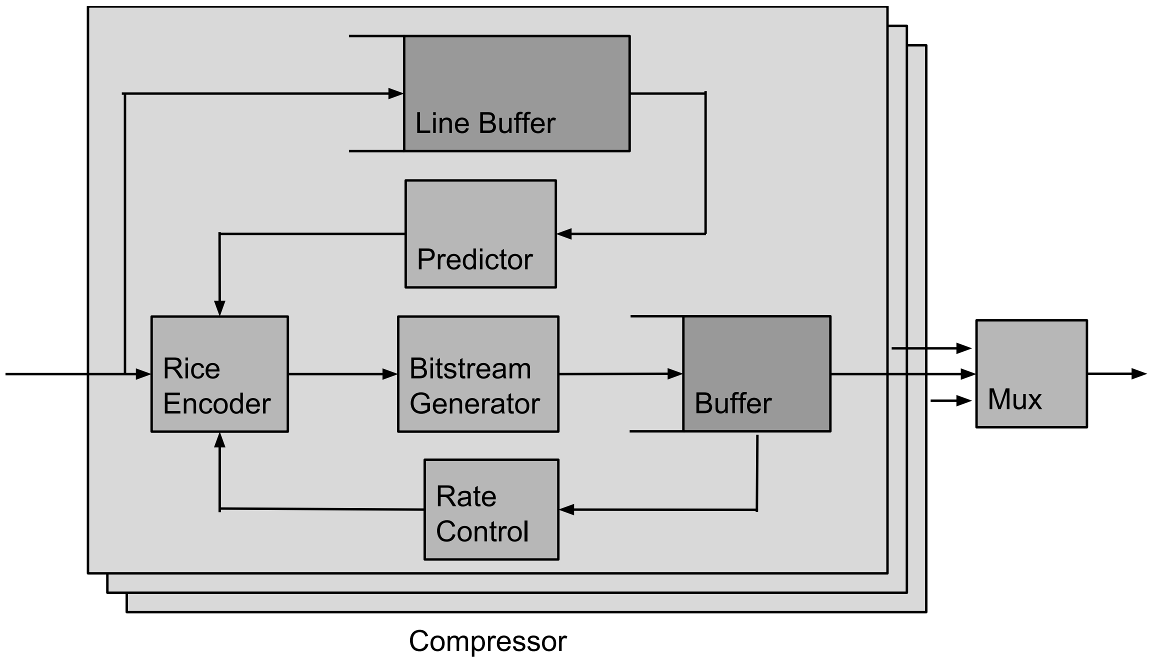

Figure 4: Architecture of the compressor.

The bitstreams are stored in a FIFO buffer. The multiplexer module reads 32-bit words from each buffer, and writes them to DRAM in correct order.

2.1.2 Decompressor Architecture

Decompressor design, shown in Fig. 5, is mostly symmetrical to the compressor. Same as in the compressor, three instances of the decompressor are used for each of the three color channels. The single bitstream that is comprised of three bitstreams is demultiplexed by the demultiplexer module. The 32-bit words are extracted from the bitstream and are pushed to their matching buffers. These words are popped from the buffer by the bitstream extractor, which in turn extracts the encoded symbols from the bitstream for the Golomb-Rice decoder to process. Golomb-Rice decoder module decodes the residual. The residual is then added to the predicted pixel to get the reconstructed pixel. The reconstructed pixels are stored in the line buffer to be used for predicting upcoming pixels.

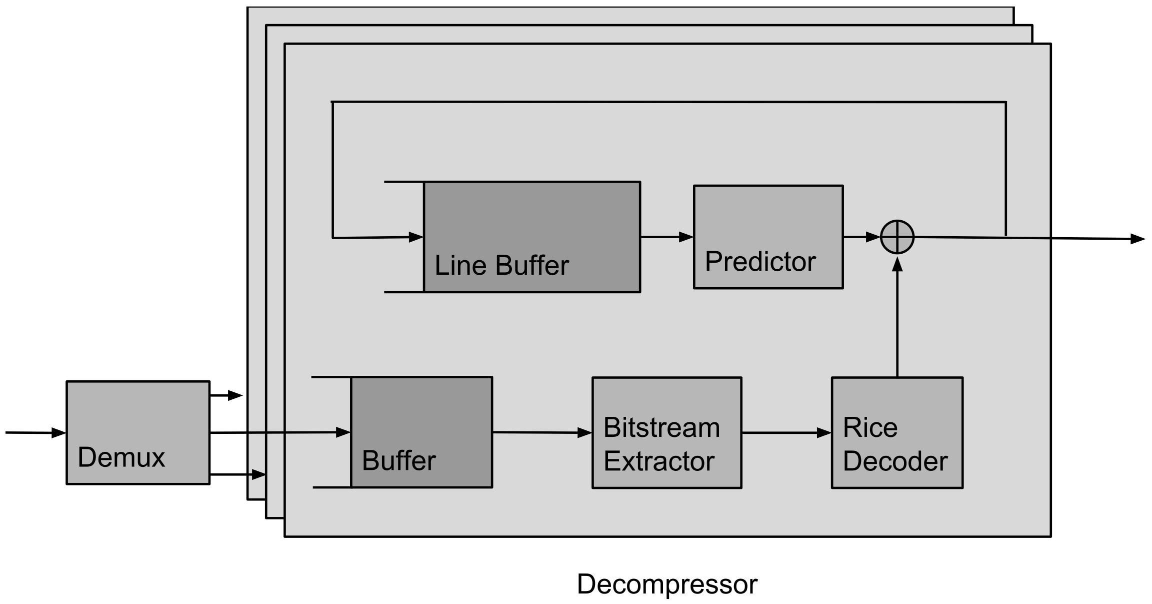

Figure 5: Architecture of the decompressor.

2.2 Quantization

The proposed compression method implements a lossy mode to cope with the situation when the bitstream buffers get filled. The quantization method quantizes the pixels by truncating the least significant bits of the pixel values, i.e. it quantizes the pixels by powers of 2. This operation is implemented with simple logic shifters. Quantization level corresponds to the number of the least significant bits truncated. and is determined by the rate control algorithm, which is discussed in Section 2.5.

On reconstruction, bits from the predicted pixel, which is always an 8-bit value, are put in place of the lost bits as shown in Fig. 6.

2.3 Prediction Method Cycling

The proposed prediction system utilizes multiple prediction methods through cycling them as shown in Fig. 7.

At periodic checkpoints, the amount of bits written is checked against predefined threshold levels. If the threshold for the current method is exceeded the prediction method gets switched. The threshold levels can be set to different levels for biasing

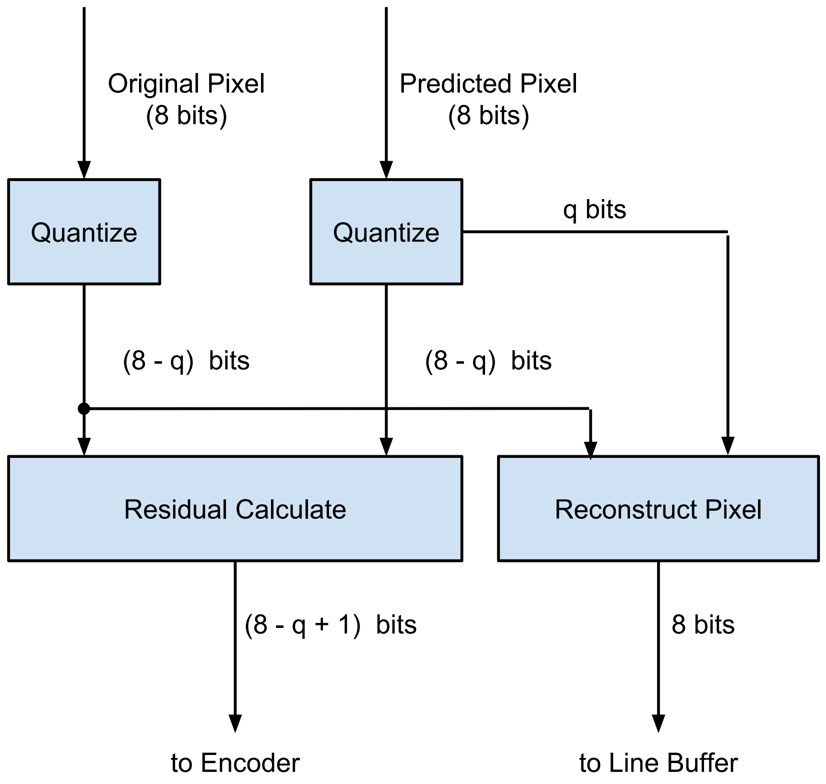

Figure 6: Quantization and reconstruction of pixels.

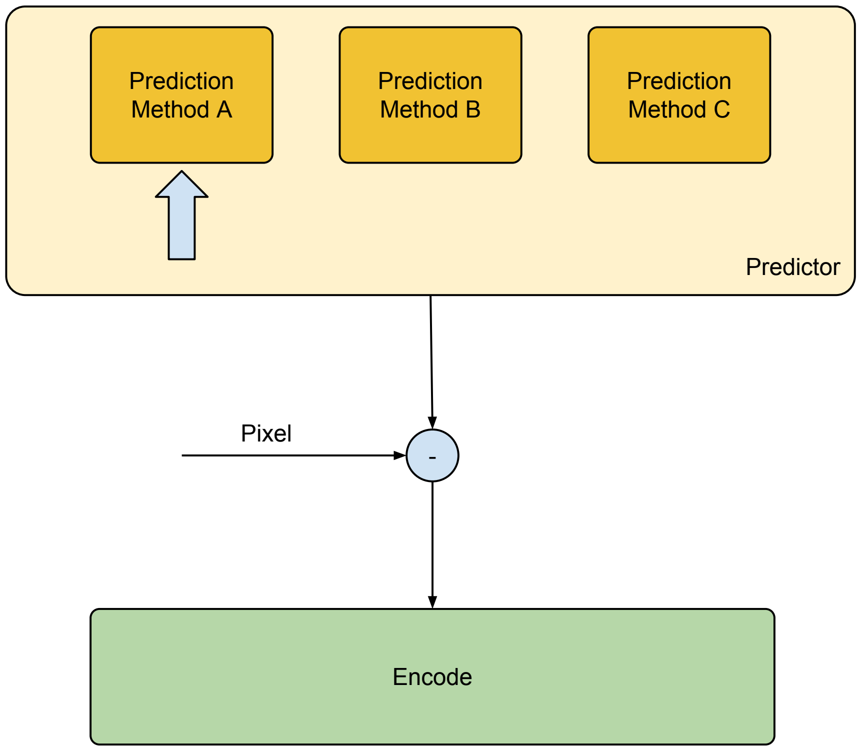

Figure 7: Prediction method cycling.

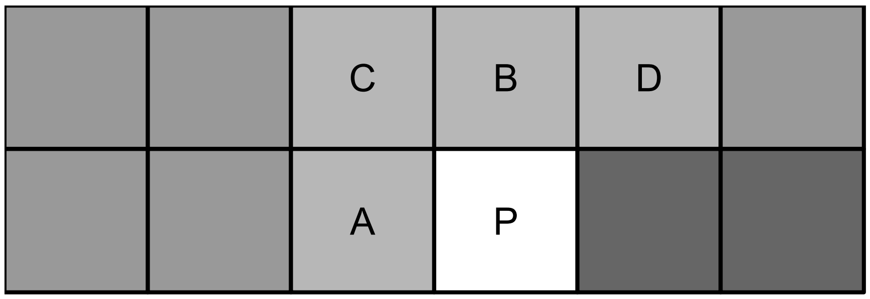

Figure 8: Pixels used for prediction.

certain methods for performance purposes. It should be noted that this algorithm does not require any additional metadata written to the bitstream. The algorithm is repeatable by the decompressor without side information.

The algorithm is as follows:

Prediction Method Cycling Algorithm

```
const METHODCNT = 2;
const FAILTHRESHOLD[METHODCNT];
const CHECKPOINT = 20;
fun PredictionMethods[METHODCNT](); //Prediction methods

extern pixel; //Pixel to be encoded
extern neighbors; //Neighboring pixels used for prediction

var pixCnt = 0;
var writeCnt = 0;
var curMethod = 0;

while (true) //Infinite loop
{
    pixCnt = pixCnt + 1;
    prediction = PredictionMethods[curMethod](neighbors);
    writeCnt = writeCnt + RiceGol(prediction, pixel);

    if(pixCnt == CHECKPOINT)
    {
        if(writeCnt >= FAILTHRESHOLD[curMethod])
        {
            curMethod = (curMethod + 1) % METHODCNT;
        }
        writeCnt = 0;
    }
}
```

Cycling prediction methods is advantageous as different areas of frames show different characteristics. An area may have horizontal patterns, while another may have vertical ones. The algorithm can be set to cycle over the pixels A, B, and D shown in Fig. 8. One more advantage of this method is that using and cycling between simpler prediction methods decrease the minimum clock period and also the footprint. One thing to note is that, for the first line of a frame since there is no previous line the prediction is always set to be pixel A.

The implementation used to obtain the results in Chapter 3 makes use of two prediction methods: JPEG-LS prediction method and predicting pixel C method. The former performs well in natural images. However, in some synthetic content, such as computer generated test patterns, checkerboard patterns may occasionally occur. In such a case, JPEG-LS prediction method fails and predicting C performs better.

JPEG-LS's prediction method's low complexity and high performance in natural images make it a suitable choice. Three neighboring pixels A, B, and C are used for prediction. As a result, only one line of pixels needs to be buffered. The JPEG-LS predictor is an edge detector. It predicts the pixel as A if the detected edge is horizontal and as B if the detected edge is vertical. Otherwise, it is predicted as A + B - C:

$$Prediction = \begin{cases} min(A,B) & \text{if } C \geq max(A,B) \\ max(A,B) & \text{if } C \leq min(A,B) \\ A+B-C & \text{otherwise.} \end{cases} \quad (1)$$

Although it is low complexity, a straightforward implementation of the JPEG-LS may become the critical path of the whole design as it burdens timing with a relatively long feedback loop that is part of the compression/decompression pipeline. The calculation of the prediction must be done in exactly one cycle, as the reconstructed pixel will immediately be used as pixel A for the prediction of the next pixel. This immediate dependency may become the critical path.

Another challenge of using the JPEG-LS prediction method is the fact that it is part of a patented compression algorithm. Being as such, it may not be suitable to use commercially without the necessary permissions. In that case, cycling over predicting the pixels A, B, and D may be a good substitute.

2.4 Golomb-Rice Coding

Golomb-Rice coding method is used to compress the residual. It is an entropy coding method that assumes a fixed geometric distribution, where higher numbers are coded with longer words. It is suitable for prediction based compression algorithms such as the proposed method.

2.4.1 Syntax

Golomb-Rice codes consist of two parts. The input integer is divided by a number that is power of 2. The remainder is written in fixed size bits as binary codes, while the quotient is coded with unary codes. A stop bit "1" separates the binary coded part from the unary coded part. Since the prediction residual is a signed integer, a sign bit is added to the coded word to indicate the sign of the residual. The length of the binary coded part is defined as the parameter *BINCODELEN*. The syntax used

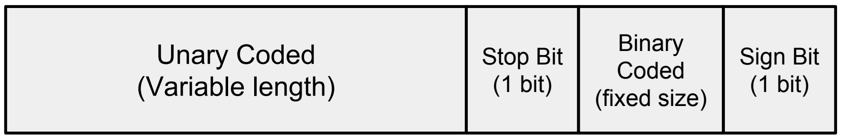

Figure 9: The Golomb-Rice code syntax used in the proposed algorithm.

is shown in Fig. 9 while the encoding process steps are shown in in Fig. 10.

2.4.2 Unary Codes

The most significant bits are encoded with unary codes. It makes sense to compress the most significant bits with unary codes as the most significant bits of the residual are usually numbers close to 0. Unary codes are of variable length and used to encode unsigned integers. ”0” bits as many as the number that is to be represented are written and a stop bit ”1” is added to the end. As such, smaller values are encoded with less number of bits.

2.4.3 Binary Code

The least significant bits of the residual are not processed and copied directly to the end of the symbol. The least significant bits of the residual have high entropy as they are highly effected by noise in the image. Therefore, they are not very suitable for compression.

2.4.4 Cutoff for Unary Codes

Although the unary codes can be arbitrarily long, handling of codes that are too long is technically challenging and it would also hurt the compression ratio. To address such issues, an upper bound for the unary coded part is introduced. A value *CUTOFF* is defined as the longest allowed unary code length. It is set to be 10 bits. In the case when the residual is high and the length of unary codes go over *CUTOFF*, *CUTOFF* number many of ”0” bits followed by the value of the quantized pixel is written as the

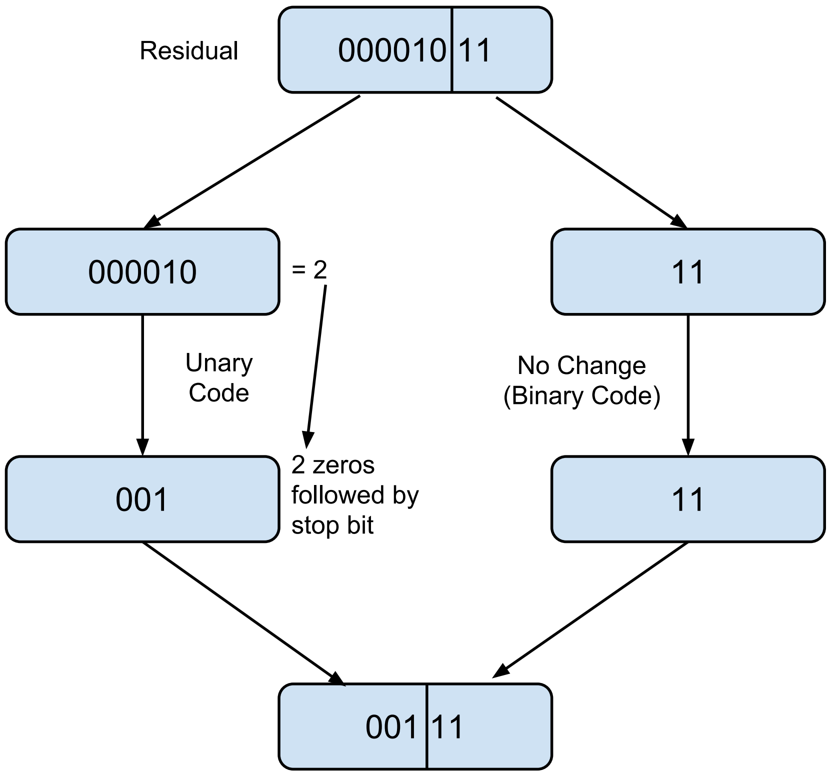

Figure 10: Golomb-Rice coding process steps.

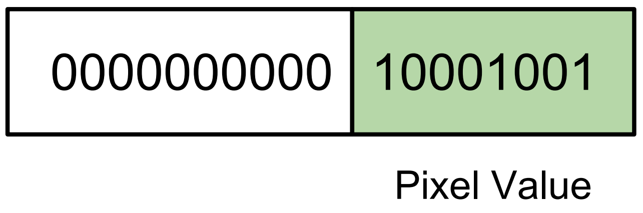

Figure 11: *CUTOFF* bits followed by the pixel value is written.

coded word as show in Fig. 11. The same syntax is also used as a part of underflow control, whenever the *NEAREMPTY* flag is set to "1", the pixel is encoded as if the residual is high. Fig. 12 shows the control flow for the encoding process.

2.4.5 Complexity and Design Decisions

The compression efficiency of Golomb-Rice codes is determined by the length of the binary coded part *BINCODELEN*. As the binary codes become longer, larger residual values are encoded with shorter symbols. However, the length of the shortest code, which is calculated by *1 (Stop Bit) + 1 (Sign Bit) + BINCODELEN*, becomes longer. The length of the shortest code determines the highest TCR achievable. Therefore, the binary code length needs to be set according to TCR.

Although it is possible to use dynamically sized binary codes instead of fixed sized codes to benefit from shortening the minimum code size; it was ultimately discarded due to complications of its implementation would introduce. The primary reason for using fixed size binary codes is the lower complexity. In the current syntax, only the unary coded part is of variable length. To obtain the length of a coded symbol only the location of the first "1" bit needs to be determined. As such, the operation of calculating the length of the coded symbols, which ultimately determines the maximum clock frequency of the whole design, in the bitstream is significantly short.

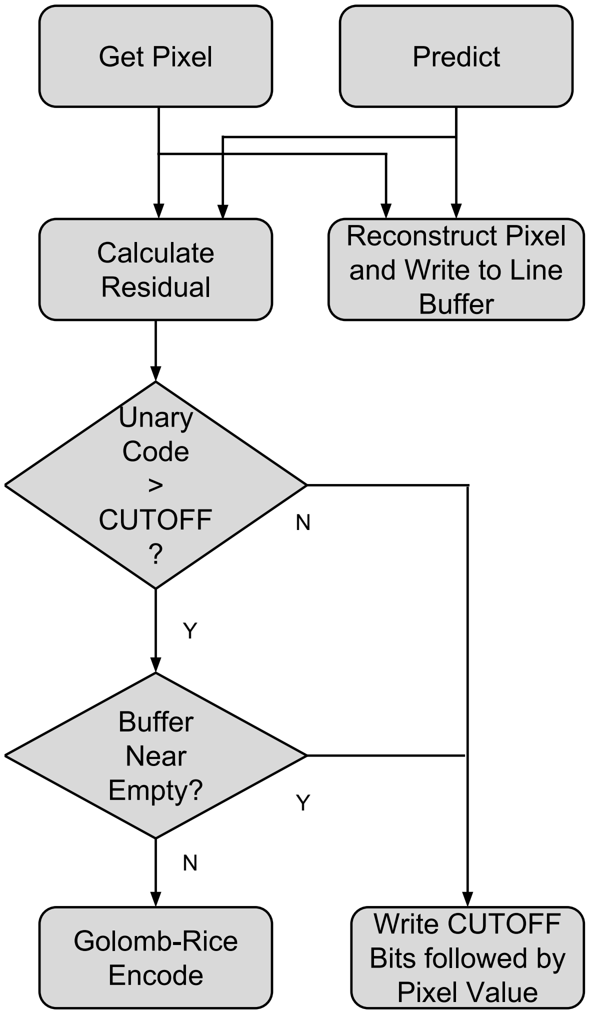

Figure 12: The algorithm for encoding of each pixel.

2.5 Rate Control

The rate control system is responsible for adjusting the rate at which the data is pushed into the bitstream buffer. The rate at which the buffer is depleted is fixed and it is determined by TCR. Rate control logic must assure that the buffer never overflows or underflows, even in the worst case scenario.

2.5.1 Underflow Prevention

There has to be meaningful data in the buffer every time a data pop request is received. This is necessary as the data words gathered from the three color channels are multiplexed into a fixed order before the bulk of data is pushed into DRAM. The data words popped from the buffer will be garbage if the buffer is in empty state.

Underflow prevention is handled by a flag raised by the rate control module. If the buffer is depleted down to a pre-determined threshold level, the rate control module sets the $NEAREMPTY$ flag to "1". Whenever the $NEAREMPTY$ flag is "1", Golomb-Rice encoder module encodes the incoming pixels as if the prediction residual was too large. In this mode, the size of the generated words are equal to *CUTOFF + 8 bits*. In this case, the buffer gets filled at a rate higher than the depletion rate. This ensures that there is always meaningful data in the bitstream buffer.

2.5.2 Overflow Prevention

Overflows are prevented by switching to lossy mode. The rate control determines the quantization level and aims to keep spikes from occurring in the quantization level. The input frame is divided into fixed size blocks of sequential pixels and the quantization level is decided at the start of each block. The block is then encoded at the decided quantization level. The quantization level is determined by the state of the buffer and is not written to the bitstream.

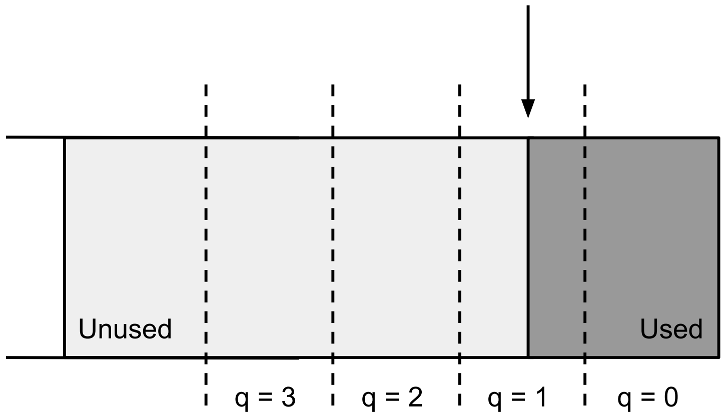

Figure 13: The quantization level decision mechanism as determined by the buffer state.

2.5.3 Quantization Level Decision

Preventing overflows and underflows is not the only objective of rate control. The rate control algorithm is designed so that the quantization level difference between two sequential blocks is never greater than 1. The reasoning is that if the quantization level change is not smooth, quantization level differences between two consecutive blocks could be so high that the resulting loss effect will be visually noticeable.

This behavior is ensured by the following algorithm. The rate control decides the quantization level by the state of the bitstream buffer. Threshold levels are defined for each possible quantization level as shown in Fig. 13. The difference between the threshold levels is calculated by taking the worst case into account, which is easily calculated by *BLOCKSIZE * (8 + CUTOFF - (TARGETBITRATE))*. Note that this is not a runtime operation. The threshold levels are defined at compile time.

The block size determines the minimum possible size of the bitstream buffers as all possible quantization levels must be valid locations in the buffer for the rate control algorithm to function as intended.

2.5.4 Overflow Failsafe Mode

The compression switches to failsafe mode if the buffer gets filled to a critical level in order to prevent the buffer from ever overflowing. The decision to go into the failsafe mode is done by checking if the highest quantization level allowed which is determined by TCR. For example, for the TCR of 1.50, the target bitrate is (8 / 1.50) = 5.33. If the quantization level determined by rate control ever goes as high as ceil(8 - 5.33) = 3, the quantized pixel value is written to the bitstream in 5 bits rather than the Golomb-Rice coded symbol. In this state, the rate at which the buffer is drained will always be more than the amount that is written, and quantization level will eventually go back to the previous level.

2.5.5 Rate Control in Decompressor

The proposed rate control algorithm in the compressor indirectly performs rate control for the decompressor by exploiting the symmetrical properties of the compressor and decompressor.

The design allows for buffer states for the compressor and decompressor to be the opposites of each other. The filling rate at the processing of a particular pixel of a frame will be equal to the depletion rate for the decompressor. Since the decompressor buffer will be filled with the fixed depletion rate used for the compressor buffer, the change in buffer states will be identical and the state will be symmetrical as shown in Fig 14.

This symmetrical property also allows the decompressor to extract the quantization level information at the start of blocks by using the same threshold levels used by the compressor rate control. Therefore, writing the quantization level information to the bitstream is not necessary.

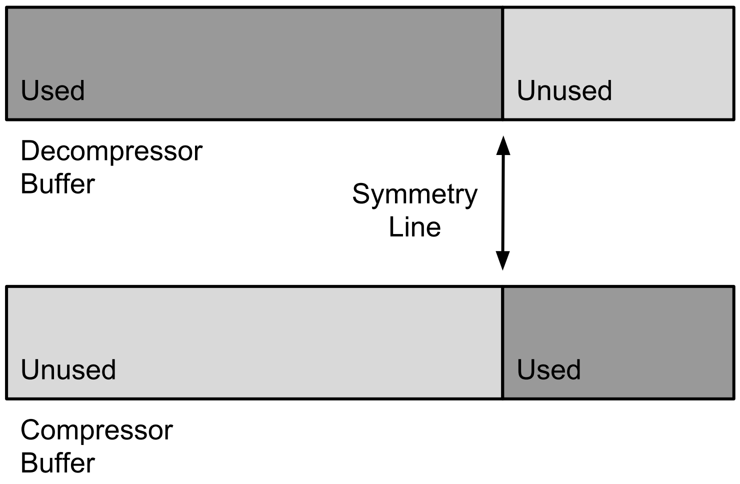

Figure 14: The buffer states are symmetrical when at the same pixel.

2.5.6 Achieving Symmetry

To achieve the symmetrical state, the initial states of the compressor and decompressor buffers must be symmetrical. Decompressor needs to pre-fill its buffer before decompression begins. Normally, the compression would begin with an empty buffer. In order to start at a symmetrical state, decompressor would need to completely fill its buffer which is the same size as the compressor buffer. There are two issues with this:

1. Starting compression with an empty buffer may result in an immediate underflow depending on the target bitrate. As a result, the draining of the buffer cannot be started immediately and needs to be delayed.

2. Decompression is done in sync with the raster scan timing. The pre-fill stage is given only 1 line of scanning time. As such, the pre-filling of the decompressor buffer must be done in a limited amount of cycles and there may not be enough cycles to completely fill the buffer if the buffer size is large.

To address the first issue, a valid solution is to start draining after a fixed number of cycles. This would ensure that the initial read will never be garbage. However, this would cause another problem regarding symmetry. The state of the buffer when draining begins will be unknown as the amount of data written in the fixed number of cycles depends on the pixel data. As such, decompressor cannot know the initial state for the symmetrical state unless it is explicitly written to the bitstream somehow.

The solution used to overcome both issues is as follows. Initially, compression starts in a non-empty buffer state with garbage data in the buffer. The number of garbage words in the initial state is a known preset value and compression starts filling its buffer with meaningful data as the garbage words are popped. The number of words popped from the buffer is counted and the garbage words are tossed and are not written to the DRAM. Since the initial state of the buffer is known, decompressor needs to fill its buffers to the exact opposite of this state.

The number of words that decompressor needs to prefetch to get to the symmetry line equals to the size of the buffer subtracted by the number of garbage words. The number of garbage words can be used to set the number of words needs to be prefetched, allowing it to match the maximum number of words decompressor can prefetch in the limited number of cycles.

2.6 Coded Symbol Concatenating and Extraction

Golomb-Rice coded symbols are of variable length. The maximum possible length of a symbol is *18 bits = CUTOFF + 8*. For the compressor, a mechanism is needed to concatenate the variable length symbol is into a continuous bitstream that is divided into chunks of 32-bit words. Similarly, decompressor needs to be able to the opposite of this operation and extract the variable length coded symbols from the continuous stream.

2.6.1 Concatenating Symbols in the Compressor

Bitstream generator handles the following:

- Concatenate variable length symbols into a single bitstream.

- Divide the resulting bitstream into 32-bit words.
- Handle cases when the coded symbols span two different 32-bit words.

The Golomb-Rice coder module has two output signals that is fed to the bitstream generator. First signal is the 18-bit data signal, the maximum length of a coded symbol, which gives the coded symbol. The second signal tells how many bits out of the 18-bit data is valid starting from the least significant bit.

A 64-bit register is used as a circular bit buffer to concatenate the variable length symbols. The two 32-bit halves of the 64-bit register are written to the bitstream buffer once they are completely filled with meaningful bits.

The concatenation operation is done with rotators and bitwise operations as shown in Fig. 15. The incoming data is rotated by the offset value so that the 64-bit rotated output has the coded symbol at the correct location. The offset to where to write the next symbol is stored in a register. The exact amount to rotate the 18-bit input is *offset + 18 - number of valid bits* as the meaningful data starts from the least significant bit. The rotated output is then used in the bitwise-or operation to concatenate the symbol to the 64-bit register. However, since the 64-bit register is used as a circular buffer there will be garbage data at the intended location and simply using bitwise-or operation won't be correct. In order to keep the operation non-destructive, a 64-bit mask is also generated with the use of a second rotator. This mask is then used as a helper to complete the concatenation operation. Whenever one of the two 32-bit halves of the register are filled with meaningful data, that 32-bit segment gets written to the buffer the next cycle as shown Fig. 16 and the 32 bits become available space to write incoming data.

2.6.2 Extracting Symbols in the Decompressor

Bitstream extractor needs to reverse the operation carried out in the bitstream generation in the compressor. The bitstream that consists of concatenated Golomb-Rice

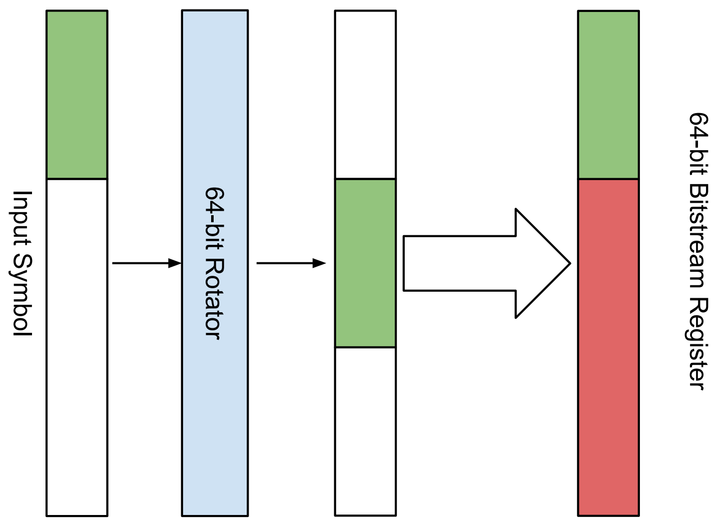

Figure 15: Symbol concatenation with the use of a rotator.

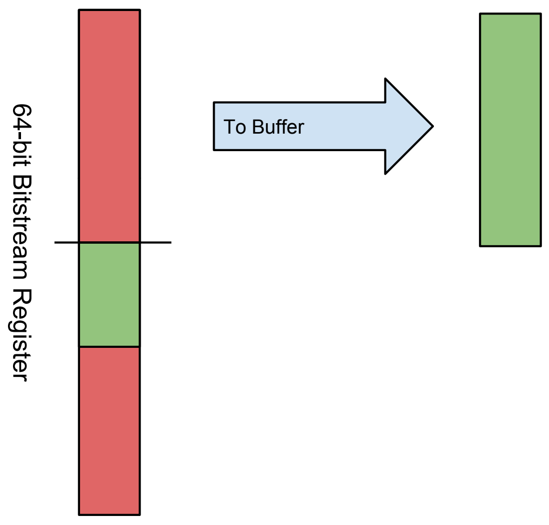

Figure 16: Extraction of 32-bit words from the 64-bit register.

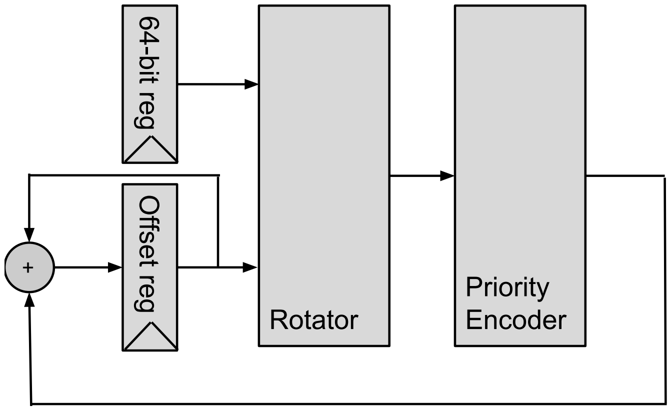

Figure 17: The critical path in decoder that determines the minimum clock frequency.

codes is fetched from the bitstream buffer as chunks of 32-bit words. Golomb-Rice codes are not specifically aligned to any part of the 32-bit words and can also span two words. Similarly, to assure that the whole symbol is always available, the upcoming two words from the bitstream buffer are fetched and stored in a 64-bit register available to the Golomb-Rice decoder. The decoder expects input that is aligned to the beginning of a coded symbol. A 64-bit rotator is used to adjust the starting bit to achieve this. Location of the starting bit is stored in an offset register and needs to be updated for the next cycle in order for the next coded symbol to be ready.

The amount to increase the offset register, the length of the coded symbol, is determined by locating the first "1" bit starting from the most significant part. The circuit that does this exact operation is known as a priority encoder. A priority encoder with input bit width *CUTOFF* is used. The amount to increase the offset register becomes *Position of the first "1" bit + BINCODELEN + 1 (Sign bit).*

Whenever 32 bits of data are processed, another 32-bit word is fetched from the bitstream buffer and pushed into the 64-bit register so that there is always data to process in the 64-bit register.

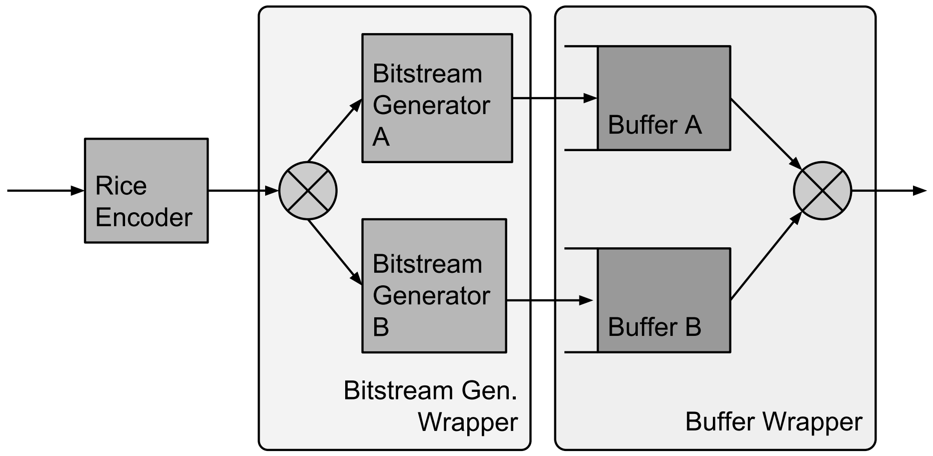

Figure 18: Dual stream generation in the compressor.

2.7 Multiple Stream Generation

The critical logic that determines the minimum clock period lies in the bitstream extraction module shown in Fig. 17. The coded symbol syntax was chosen to be as simple as possible to make the symbol length calculation a short operation. However, even with only one variable length calculation, as it is chained after rotator and adder circuits, the minimum clock period is still heavily impacted. The synthesized design was not able to meet the 150 MHz clock frequency target on Xilinx Spartan-6 FPGA with this structure. Dividing this logic into two stages is not a direct solution, as that would halve the pixel throughput by stalling the pipeline.

To meet the throughput and timing constraints, a dual-bitstream approach was taken. This approach makes it possible to divide the offset register increment operation into two stages by duplicating a relatively small part of the hardware in both the compressor and decompressor.

2.7.1 Design Changes in the Compressor

The bitstream generators and the buffers are duplicated as shown in Fig. 18. Each cycle, the bitstream pointer is switched and the generated symbol is written to the pointed bitstream. As a result, the coded symbols for even and odd numbered pixels are pushed into separate buffers. The outgoing words are popped from buffers A and B in order.

The bitstream generators and the buffers are put in wrapper modules that keep the existing input/output interfaces to other modules. Therefore, encoder and multiplexer modules are not effected by the introduction of the second bitstream.

There are some minor changes made to the rate control module. Since there are now two output buffers, rate control needs to track two buffers. Thus, there are two quantization levels for the two streams. This in turn means that odd and even numbered pixels may be quantized at different levels.

2.7.2 Design Changes in the Decompressor

Bitstream extractors and the buffers are duplicated similarly to the compressor. The input to Golomb-Rice decoder is switched each cycle and a symbol is processed from that bitstream. Similarly, wrapper modules are used to keep the input/output signals. Consequently, Golomb-Rice decoder is not modified as it is unaware of the changes and since the input signals do not change.

Since output required from the wrapper module is one symbol per cycle, to meet the output rate with two actual bitstream extractors the rate required becomes one symbol per two cycles. As such, the critical path can be divided into two cycles.

2.8 Multiplexing and Demultiplexing

Memory bandwidth output at the targeted ratio is provided with a token bucket algorithm implementation. The target bitrate is represented in an 8-bit fixed point value with 4 bits of integer and 4 bits of fraction. Each cycle, a counter is incremented by the targeted bitrate value. When the counter goes over the word size of 32, a word is popped from the buffer and the counter is decremented by 32.

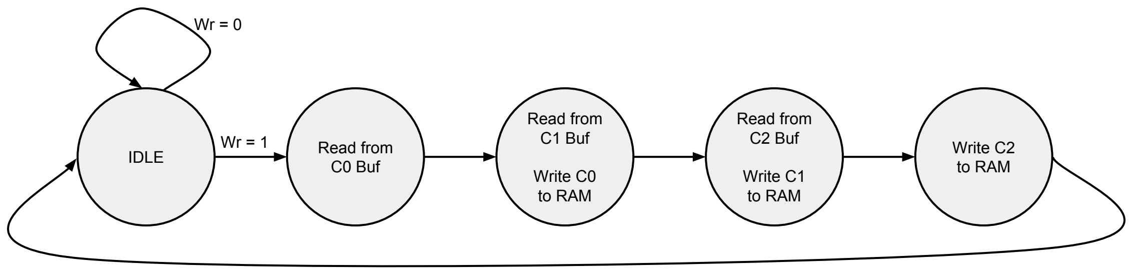
Wr = 0
IDLE
Wr = 1
Read from C0 Buf
Read from C1 Buf Write C0 to RAM
Read from C2 Buf Write C1 to RAM
Write C2 to RAM

Figure 19: State machine for multiplexing at equal bitrates.

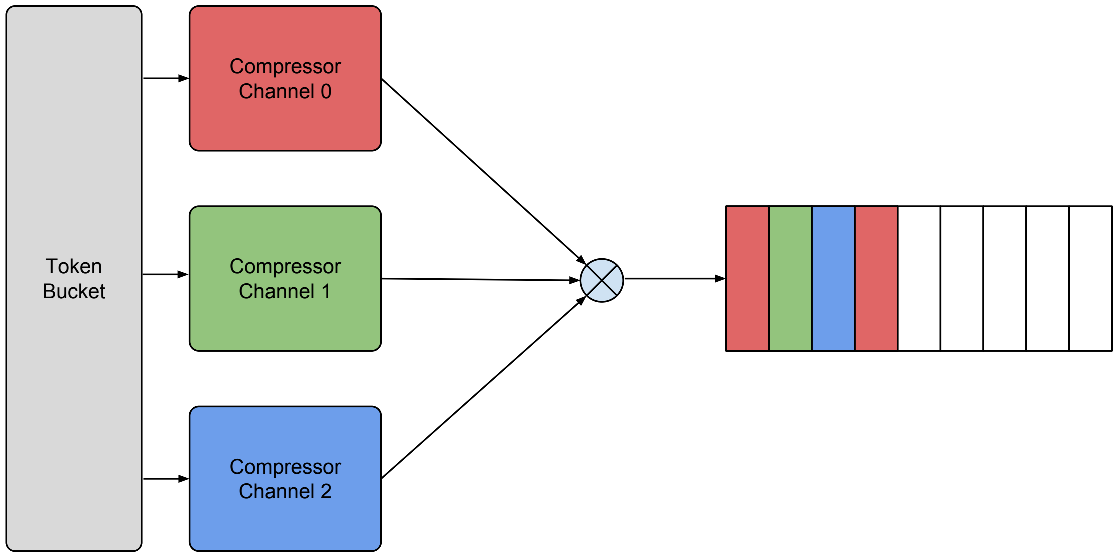

Figure 20: Simple ordering when equal target bitrates are used.

2.8.1 Equal Bitrates for Three Channels

Multiplexing of the words from the three channels is mostly trivial when the color channels are compressed to the same target ratio. One token bucket module is used for three channels. When a read operation is requested, one word is read from each color channel's bitstream buffer. When they are written to DRAM, the words are ordered in the repeating order of C1, C2, C3. The simple state machine for the behavior is shown in Fig. 19 where the signal *Wr* is the signal from the token bucket module that indicates the start of the writing process. Since the resulting word ordering is fixed as shown in Fig. 20, no tagging is needed to the words.

2.8.2 Unequal Bitrates for Three Channels

In the YCbCr color space, the human eye is more sensitive to the Y channel and almost all video content is in YUV4:2:0 - the Y channel is sampled in 4 times the sampling rate of the Cb and Cr channels. Generally, giving more bandwidth to the Y channel significantly increases the perceived image quality.

In order to benefit from giving more bandwidth to the Y channel it was necessary to add support for setting different bitrate targets to different channels. To add this functionality to the proposed design, a more complex multiplexer shown in Fig. 21 was implemented with priority queues. As the bitrate targets are different, in this scenario three token bucket modules are used. The three token buckets run in parallel and whenever one raises the flag that indicates it is ready to push data, one word is popped from that buffer. When two token buckets raise their flags in the same cycle they get popped together and the one with higher priority gets written out to DRAM directly, while the word from the lower priority buffer is put in a queue in the multiplexer until the next cycle. If all three flags are raised in the same cycle, it is handled similarly, except that the third word is stalled for two cycles. The queue that provides this functionality has a depth of two words and can accept two writes in one cycle – its implementation is trivial.

It should be noted that this algorithm also results in a deterministic word ordering and is repeatable by the decompressor; therefore no tagging is needed.

2.9 Compression Parameters

There are several compile time parameters that determine the performance of the resulting synthesized design. The most important parameters are the bitrate targets for individual channels. The second most important parameters are the *BINCODELEN* values. Since these values determine the minimum length a coded symbol can have, they effect the performance of the compression directly. These value need to be set

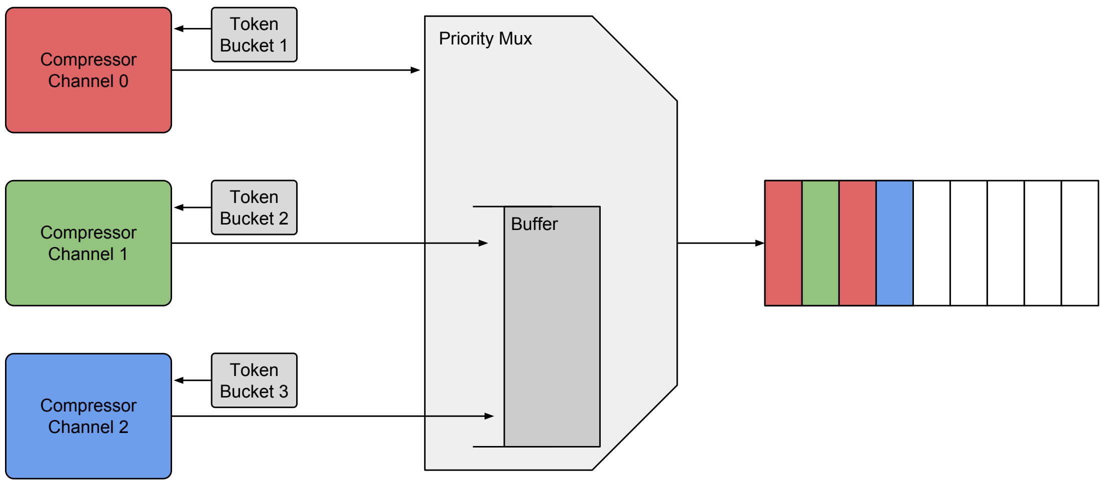

Figure 21: Priority multiplexing with three token bucket modules.

according to the target bitrate for each channel.

2.10 Y Bypass Mode

There may be cases where the required TCR level is a low value such as 1.25. For such scenarios if the color channel used is YCbCr, a mode where Y channel data is not compressed but inserted into the bitstream as raw data is defined.

This has the following advantages:

- Y channel, the channel the human eye is more sensitive to, will always be lossless.
- Rather than using entire compressor and decompressor modules for the Y channel, only a module that buffers the Y channel and inserts it into the bitstream is required. Since modules such as rate control, Golomb-Rice encoder and predictor are not used, the occupied chip area will be reduced.

In this scenario, the Cb and Cr channels will be compressed more aggressively to to compensate the high bandwidth given to the Y channel. However, the compression performance of the algorithm proves to be good enough to achieve this.

CHAPTER III

RESULTS

This chapter presents synthesis results and compression performance evaluation test results of the proposed design and compares them to the results taken from [9]. Section 3.1 presents FPGA and ASIC synthesis results and Section 3.2 presents compression performance evaluation test results.

3.1 Synthesis Results

The proposed design was synthesized with both FPGA and ASIC synthesis tools. The results are shown in Table 1 and Table 2, respectively.

The FPGA synthesis results show that the design takes as low as 15% of the available resources of the targeted FPGA chip. With the maximum clock frequency achieved it is able to process 1920x1080 resolution frames at 60 Hz.

The ASIC synthesis results are compared to the results taken from [9]. The results in Table 2 show that when similar configuration is used the proposed design is able to achieve clock frequencies 95% faster. With the achieved clock frequency, the proposed design is able to achieve almost the same pixel throughput without the need for multiple compressor and decompressor cores. The proposed design also has

Table 1: FPGA synthesis results

Target FPGA Chip	Xilinx Spartan-6 XC6SLX45
Synthesis Tool	Xilinx ISE 14.6
Max Clock Freq.	154 MHz
LUT Utilization	4272/27288 (15%)
Block RAM Utilization	14/116 (12%)

Table 2: ASIC synthesis results comparison

	Proposed	Proposed	[9]
Stdcell Library	TSMC 180nm Slow	TSMC 180nm Typical	TSMC 180nm Typical
Wire Load	wl10	wl10	wl10
Max Clock Freq.	235 MHz	391 MHz	200 MHz
Logic Gate Count	36k	35k	45k

Table 3: Compression configurations used in tests

Content	TCR = 2.0	TCR = 2.5	TCR = 3.0
Block Size (pixels)	60	60	60
Bitrate Y (bits per pixel)	5.62	4.18	2.69
Bitrate Cb (bits per pixel)	3.19	2.69	2.69
Bitrate Cr (bits per pixel)	3.19	2.69	2.69
Binary Code Size Y (bits)	2	1	0
Binary Code Size Cb (bits)	1	0	0
Binary Code Size Cr (bits)	1	0	0

a smaller area footprint of about 35k logic gates which is 77% of [9].

3.2 Performance Evaluation

A software simulator for the proposed compression algorithm was developed for verification, easy testing and performance evaluation purposes. The performance of the algorithm is tested in two different scenarios: lossy and lossless video.

3.2.1 Compression Parameter Settings

The software simulator of the algorithm was used to run performance evaluation tests at three different TCR settings. The configurations used for the TCR values are shown in Table 3.

3.2.2 Video Content

For compression performance evaluation tests four different video segments with different characteristic were used:

1. Bluesky (Fig. 22): In this video segment, the camera starts with a view of the sky and it pans to left as tree branches and leaves enter the view. The dark colored trees branches and leaves on a bright background create a lot of edges and high frequency changes.

2. Rush (Fig. 23): This video segment shows cars in a busy street. There are a lot of edges around the cars in the view.

3. Station2 (Fig. 24): This video segment is shot of a view from a train station. The train tracks in the view create a lot of vertical edges.

4. Sunflower (Fig. 25): In this video, a close up of a sunflower with a bee on it is shown. The detailed texture of the sunflower view generates a high frequency signal in both horizontal and vertical directions.

3.2.3 Lossy Input Tests

Most EC methods are designed to work with lossy content as they are meant to be used in video codec hardware. In this scenario, the performance of the algorithm is measured by the Peak Signal to Noise Ratio (PSNR) loss metric, which is used as a way to measure the distortion added by applying EC to the already lossy input. PSNR loss metric is obtained by subtracting Final PSNR from Base PSNR shown in Fig. 26.

Although the proposed method is not meant for only lossy video content, these tests are used to evaluate the performance of the algorithm in the most common use case scenario for HDTVs – lossy video (Blu-Ray video, DVB, etc.).

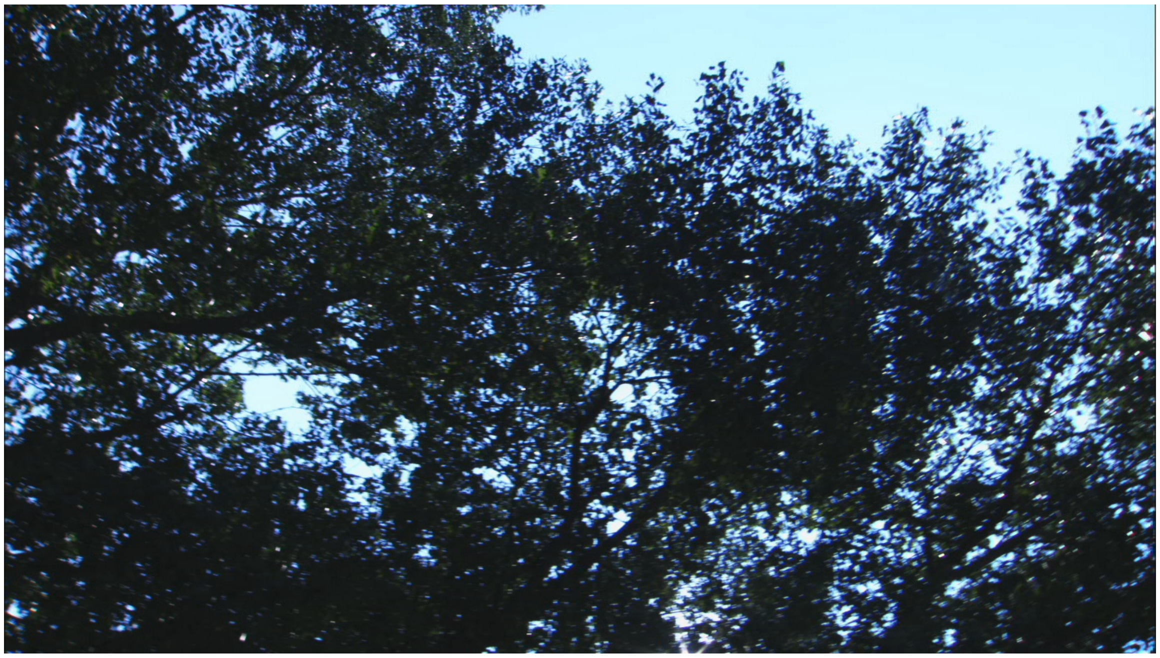

Figure 22: A frame from the video segment Bluesky.

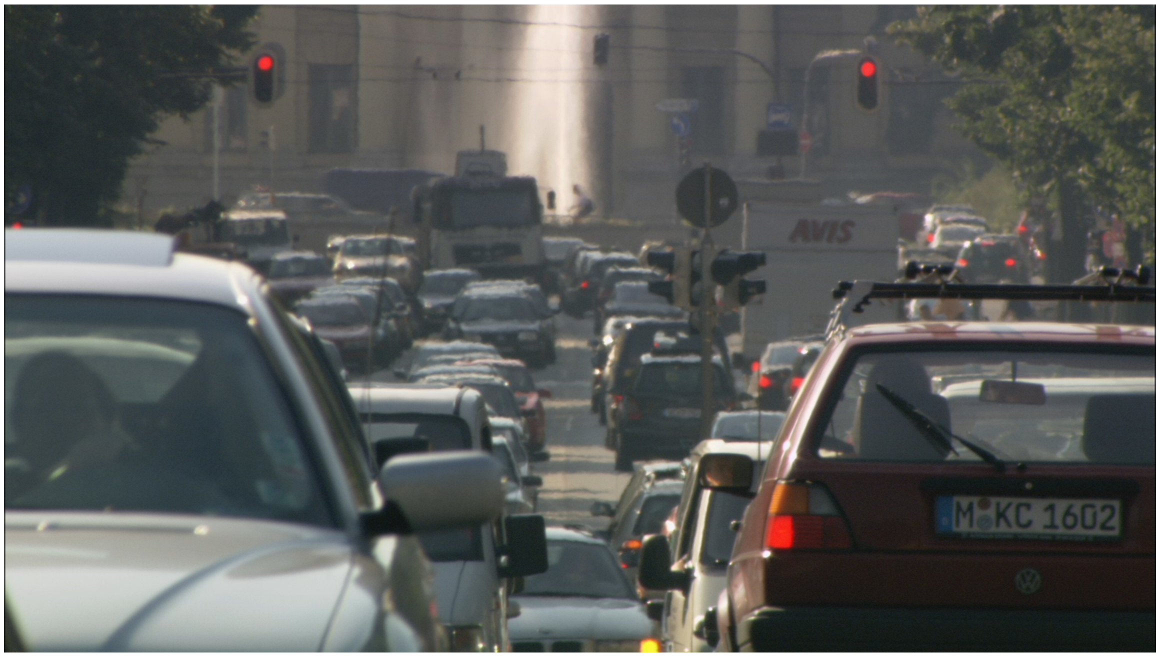

Figure 23: A frame from the video segment Rush.

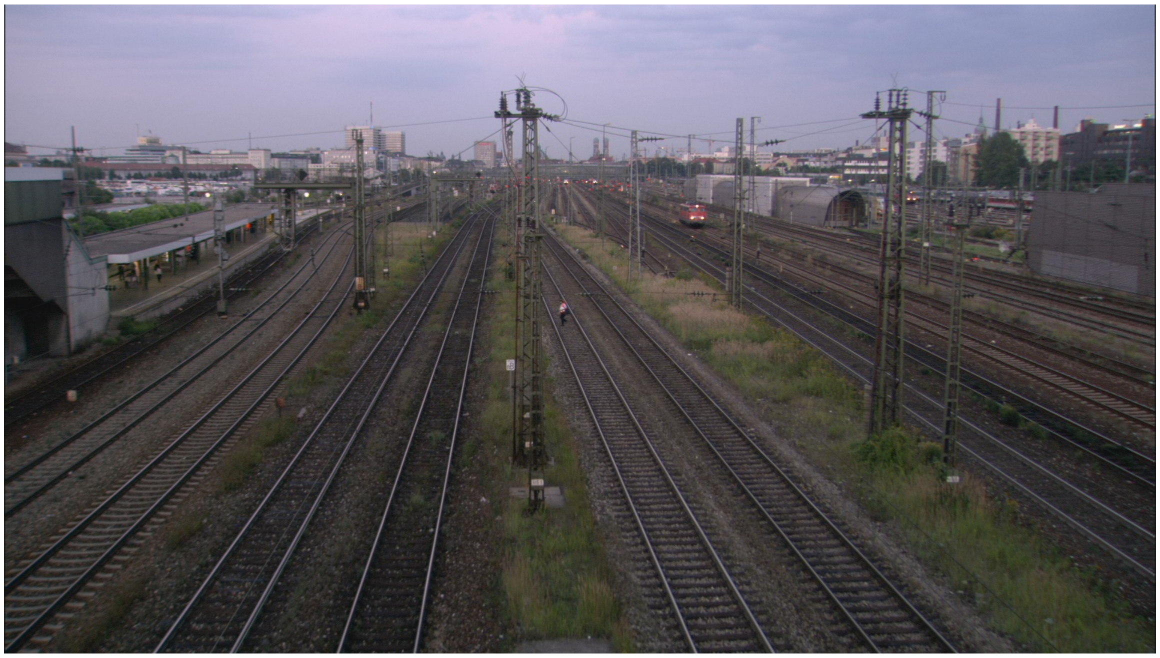

Figure 24: A frame from the video segment Station2.

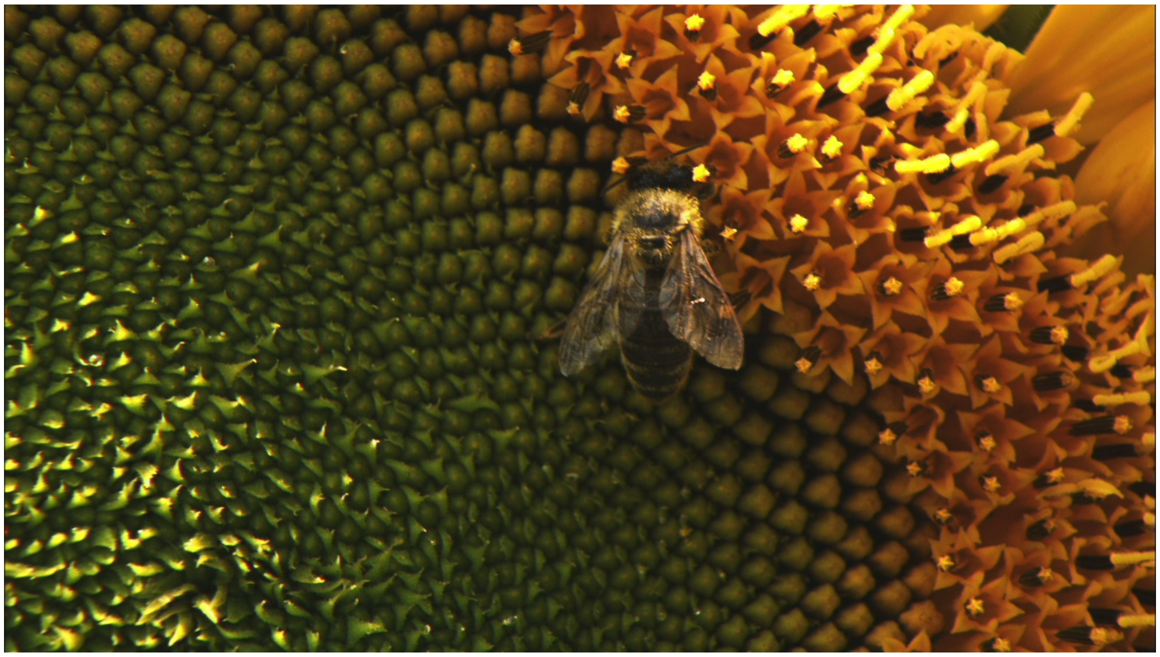

Figure 25: A frame from the video segment Sunflower.

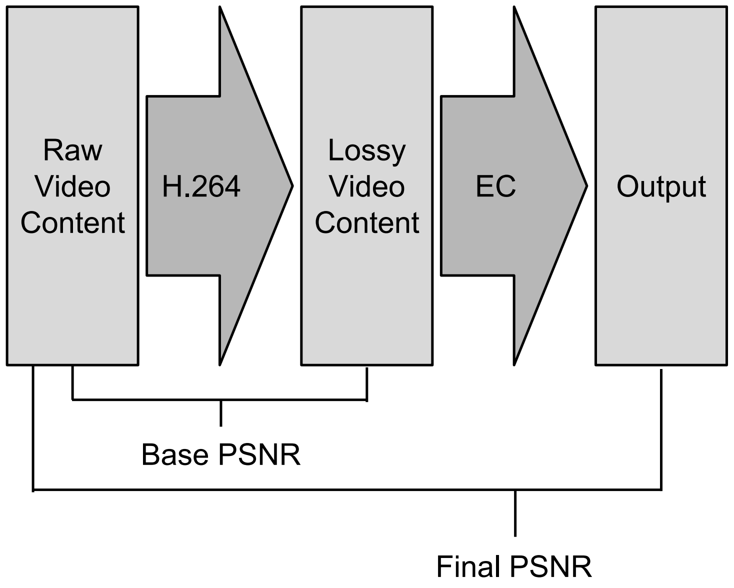

Figure 26: The PSNR values used in PSNR loss metric calculation are shown.

In this test, the originally lossless video segments are compressed beforehand at three different Quantization Parameter (QP) levels with a H.264 video encoder. The three QP levels used are 15, 20, and 25 for representing the quality levels high, medium, and basic, respectively. The results from these test are shown in Table 4. In this table, the PSNR values are the Base PSNR values shown. The PSNR loss values are compared to the ones taken from [9].

Despite the higher clock speed and smaller footprint of the proposed algorithm, the results show that the image quality achieved is at comparable levels. On the average, when compared to the results from [9] the proposed algorithm performs better at TCR 2.0 and 2.5. However, at TCR 3.0, it only performs better at QP 15. This is mostly due the fact that at TCR levels as high as 3.0, the minimum coded symbol length limits the performance.

3.2.4 Lossless Input Tests

In this test, the algorithm is run at three different TCRs on four different pieces of video content. The content is in uncompressed raw YUV4:2:0 format. The loss

Table 4: PSNR loss with H.264 video input

(High Quality QP = 15)							
		TCR = 2.0		TCR = 2.5		TCR = 3.0	
Content	Base PSNR (dB)	Proposed	[9]	Proposed	[9]	Proposed	[9]
Bluesky	48.06	0.01	0.01	0.15	0.24	3.38	5.17
Rush	48.41	0.00	0.01	0.00	0.28	1.83	5.56
Station2	47.25	0.00	0.01	0.01	2.78	2.55	9.08
Sunflower	47.26	0.00	0.01	0.00	0.11	1.05	2.49

(Medium Quality QP = 20)							
		TCR = 2.0		TCR = 2.5		TCR = 3.0	
Content	Base PSNR (dB)	Proposed	[9]	Proposed	[9]	Proposed	[9]
Bluesky	44.35	0.00	0.01	0.05	0.01	1.53	0.08
Rush	45.19	0.00	0.01	0.00	0.01	0.23	0.01
Station2	43.87	0.00	0.00	0.00	0.04	0.80	2.72
Sunflower	44.89	0.00	0.01	0.00	0.01	0.37	0.37

(Basic Quality QP = 25)							
		TCR = 2.0		TCR = 2.5		TCR = 3.0	
Content	Base PSNR (dB)	Proposed	[9]	Proposed	[9]	Proposed	[9]
Bluesky	41.21	0.00	0.01	0.02	0.01	0.69	0.28
Rush	44.14	0.00	0.01	0.00	0.01	0.03	0.01
Station2	42.40	0.00	0.01	0.00	0.01	0.37	0.38
Sunflower	43.10	0.00	0.01	0.00	0.01	0.18	0.10

is represented in dB PSNR. Although lossless video is not a common use case for consumer targeted TVs, this test is used as a way to show the performance bound of the proposed algorithm.

As shown by the results in Table 5, 3 out of 4 of the video content are compressed lossless at TCR 2.0. In the test with the video segment Bluesky, only 44 out of 216 frames are compressed with loss, and the PSNR average of the lossy frames is 68.38 dB.

At TCR 2.5, 2 out of 4 of the video segments are still compressed lossless. While only 129 frames out of 312 frames are compressed with loss on Station2, all of the frames in Bluesky are compressed lossy. The average dB PSNR values for the lossy frames are 69.79 and 65.53 respectively, which are still at near-lossless levels.

At TCR 3.0, all of the frames from all 4 video segments are compressed lossy. The average dB PSNR for all 4 video segments is 49.40. At this level, the compression artifacts may be visible to the average viewer.

Table 5: Lossless video content performance

Content	Total Frames	Lossy Frames	Avg. PSNR
(TCR = 2.0)			
Bluesky	216	44	68.38 dB
Rush	499	0	N/A
Station2	312	0	N/A
Sunflower	499	0	N/A
(TCR = 2.5)			
Content	Total Frames	Lossy Frames	Avg. PSNR
Bluesky	216	216	65.53 dB
Rush	499	0	N/A
Station2	312	129	69.79 dB
Sunflower	499	0	N/A
(TCR = 3.0)			
Content	Total Frames	Lossy Frames	Avg. PSNR
Bluesky	216	216	47.17 dB
Rush	499	499	50.67 dB
Station2	312	312	47.90 dB
Sunflower	499	499	51.87 dB

CHAPTER IV

FUTURE WORK

There are several opportunities for improvement to the proposed method which are discussed in this chapter. These possible improvements are footprint reduction, parallelization to increase the throughput, and adding random access support.

4.1 Area Improvements

Two possible ways to reduce the footprint of the design are discussed in this section.

4.1.1 Area Improvement by Sharing One Predictor

In the current design, the three compressors work independently. None of the resources are shared except for the line buffer and there is no communication in between. Although it is not possible to share most resources it should be possible to reduce the number of predictors to one. Since the color channel data is correlated spatially, it should be possible to use only one predictor and share the direction of the predicted pixel among channels. This would decrease the footprint of the whole design with little to no impact to the compression performance.

4.1.2 Using Only One Channel of Pixel Buffer

In order to support use case scenarios where the resources are even more limited, it is possible to buffer only the Y channel and not use vertical prediction on Cb and Cr channels. This would dramatically save resources on the target chip. However, it could impact the performance of the compression in certain scenarios, especially if the TCR is high.

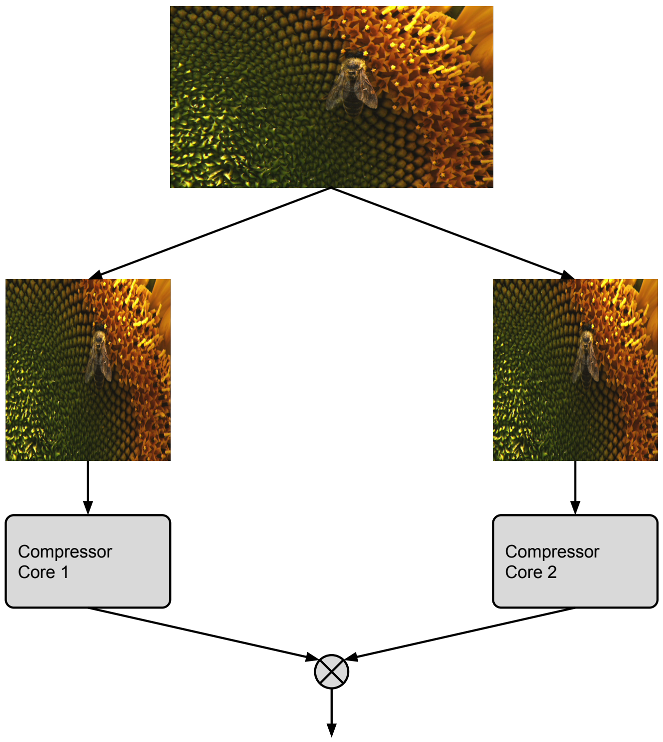

Figure 27: Parallelization with two compressor cores.

4.2 Parallelization

The FPGA implementation of the proposed design can compress 1920x1080 resolution at 60 Hz refresh rate. However, the proposed design has a processing throughput of 1 pixel per cycle. The design can be further modified to support multiple compressor and decompressor cores to achieve twice or more times the throughput by introducing parallelization. With twice the throughput, an FPGA implementation on the same chip with two cores could handle 4K Ultra HD resolution (3840x2160) at 30 Hz refresh rate. In order to support parallelization, there needs to be no dependency between data processed by the cores. Assuming 2 pixels are received each cycle instead of 1 pixel, a simple strategy for parallelization is dividing the pixels into odd and even numbered pixel streams and feeding the pixel streams to their respective compressor cores. In this scenario, since each core has its own separate line buffer and uses those pixel data for prediction there will not be any dependency between predictions for consecutive pixels. Basically, each core will be compressing a subframe of half the horizontal resolution of the full frame as shown in Fig 27. However, since the horizontal distance between pixels processed by a core will be doubled, prediction accuracy will likely be impacted. It is necessary to update the prediction method to bias towards vertical predictions as the vertical distance will not be doubled. The bit-streams from the cores can be multiplexed and demultiplexed with the same strategy used to generate multiple streams explained in Section 2.8.

4.3 Random Access

Another possible improvement is adding support for random access. The fixed TCR property of the proposed method can be further exploited to align the start of lines to fixed addresses in DRAM, therefore allowing access from the start of a line. However, the current design has dependency between lines as the prediction method requires access to the previous line. This means that in order to access a particular pixel, the frame needs to be decoded starting from the first pixel of the first line of the frame.

The frame could be divided into vertical blocks of lines that are independently compressed. This would mean disabling vertical prediction for the first line of the each block. With this feature the maximum dependency for random access could be narrowed down to the height of the vertical blocks.

CHAPTER V

CONCLUSION

In this thesis, an FPGA and ASIC compatible near-lossless image compression algorithm for raster scan input and output was proposed. The algorithm can do lossless compression of most video content at TCR 2.0 and is able to maintain near-lossless quality levels at up to TCR 2.5.

An implementation of the algorithm was realized on a low cost Xilinx Spartan-6 FPGA and can handle Full-HD (1920x1080) resolution frames at 60 Hz refresh rate while utilizing only 15% of the chip resources. On a 180nm ASIC, the achieved throughput is able to handle 4K Ultra HD (3840x2160) at 30 Hz refresh rate. The achieved pixel rate per core is higher, the footprint of the design is lower compared to similar work, and the visual quality performance of the algorithm is at comparable levels. Although currently not implemented, the proposed design can be further enhanced to support semi-random access and multiple cores.

Bibliography

[1] M. Weinberger, G. Seroussi, and G. Sapiro, "The LOCO-I lossless image compression algorithm: Principles and standardization into JPEG-LS," *IEEE Trans. Image Process.*, vol. 9, no. 8, pp. 1309–1324, 2000.

[2] X. Li, X. Chen, X. Xie, G. Li, L. Zhang, C. Zhang, and Z. Wang, "A low power, fully pipelined JPEG-LS encoder for lossless image compression," in *IEEE International Conference on Multimedia and Expo*, 2007.

[3] M. Papadonikolakis, V. Pantazis, and A. P. Kakarountas, "Efficient high-performance ASIC implementation of JPEG-LS encoder," in *Design, Automation & Test in Europe Conference & Exhibition*, 2007.

[4] H. Daryanavard, O. Abbasi, and R. Talebi, "FPGA implementation of JPEG-LS compression algorithm for real time applications," in *Iranian Conference on Electrical Engineering*, 2011.

[5] J. Kim and C.-M. Kyung, "A lossless embedded compression using significant bit truncation for HD video coding," *IEEE Trans. Circuits Syst. Video Technol.*, vol. 20, no. 6, pp. 848–860, 2010.

[6] C.-H. Son, J.-W. Kim, S.-G. Song, S.-M. Park, and Y.-M. Kim, "Low complexity embedded compression algorithm for reduction of memory size and bandwidth requirements in the JPEG2000 encoder," *IEEE Trans. Consum. Electron.*, vol. 56, no. 4, pp. 2421–2429, 2010.

[7] W.-Y. Chen, L.-F. Ding, P.-K. Tsung, and L.-G. Chen, "Architecture design of high performance embedded compression for high definition video coding," in *IEEE International Conference on Multimedia and Expo*, 2008.

[8] T. Y. Lee, "A new frame-recompression algorithm and its hardware design for MPEG-2 video decoders," *IEEE Trans. Circuits Syst. Video Technol.*, vol. 13, no. 6, pp. 529–534, 2003.

[9] T.-H. Tsai and Y.-H. Lee, "A 6.4 Gbit/s embedded compression codec for memory-efficient applications on advanced-HD specification," *IEEE Trans. Circuits Syst. Video Technol.*, vol. 20, no. 10, pp. 1277–1291, 2010.

VITA

Name: Okan Palaz

Date of Birth: 21/11/1988

Languages: Turkish, English

Education

- MS: Özyeğin University - Computer Science - 2014
- BS: Bahçeşehir University - Computer Engineering - 2011
- High School: Gaziantep Vehbi Dinçerler Fen Lisesi

Work Experience

- AirTies Wireless Networks
  Systems Engineer
  2013 - Ongoing
- Vestek R&D Corp.
  Part-time FPGA Design Engineer
  2010 - 2011

Publications

- O. Palaz, H.F. Ugurdag, B. Kertmen, Ö. Özkurt, İ. Gerçek, and F. Dönmez HDTVler İçin Yüksek Performanslı Düşük Karmaşıklık Seviyeli Gömülü Sıkıştırma Metodu, ELECO 2014, Submitted.
- S. Gören, H.F. Ugurdag, O. Palaz, Defect-Aware Nanocrossbar Logic Mapping through Matrix Canonization using Two-Dimensional Radix Sort, ACM Journal on Emerging Technologies in Computing Systems, vol. 7,

no. 3, August 2011.

- S. Gören, H.F. Ugurdag, and O. Palaz, Defect-Tolerant Logic Mapping for Nanocrossbars Based on Two-Dimensional Sort, Proceedings of 25th International Symposium on Computer and Information Sciences (ISCIS), (Lecture Notes in Electrical Engineering 62), pp. 399-404, London, UK, September 2010.
- S. Gören, H.F. Ugurdag, O. Palaz, Defect-Aware Nanocrossbar Logic Mapping using Bipartite Subgraph Isomorphism & Canonization, Proceedings of IEEE European Test Symposium (ETS), p. 246, Prague, Czech Republic, May 2010.