12-2021

ROI-AWARE CONTENT-ADAPTIVE VIDEO STREAMING SYSTEM FOR POWER SAVINGS

William Oswald
University of South Alabama, wdo1621@jagmail.southalabama.edu

12-2021

ROI-AWARE CONTENT-ADAPTIVE VIDEO STREAMING SYSTEM FOR POWER SAVINGS

William Oswald
University of South Alabama, wdo1621@jagmail.southalabama.edu

THE UNIVERSITY OF SOUTH ALABAMA
COLLEGE OF ENGINEERING

ROI-AWARE CONTENT-ADAPTIVE VIDEO STREAMING SYSTEM FOR POWER SAVINGS

BY

William Oswald

A Thesis

Submitted to the Graduate Faculty of the
University of South Alabama
in partial fulfillment of the
requirements for the degree of

Master of Science

in

Electrical Engineering

December 2021

Approved:	Date:
Chair of Thesis Committee: Dr. Na, Gong	10/25/2021
Mohamed Shaban Committee Member: Dr. Mohamed Shaban	10/25/2021
Committee Member: Dr. Kari Lippert	19 Oct 2021
Chair of Department: Dr. Hulya Kirkici	10/19/2021
Director of Graduate Studies: Dr. Robert Cloutier	10/25/2021
Dean of the Graduate School: Dr. J. Harold Pardue	11/5/2021

ROI-AWARE CONTENT-ADAPTIVE VIDEO STREAMING SYSTEM FOR POWER SAVINGS

A Thesis

Submitted to the Graduate Faculty of the
University of South Alabama
in partial fulfillment of the
requirements for the degree of

Master of Science

in

Electrical Engineering

by
William Oswald
B.S., University of South Alabama, 2020
December 2021

ACKNOWLEDGEMENTS

Before any other, I would like to thank Dr. Na Gong; without her guidance and support, this work would not have been possible. I would also like to thank my lab mates—especially Dr. Ali Haidous, Dr. Hritom Das, as these two along with Dr. Na Gong have contribute and created many sections of the IEEE journal paper which became this thesis. Thank you for supporting me and my work. I would also like to thank the members of my thesis committee, Dr. Mohamed Shaban and Dr. Kari Lippert. Thank you for the time and effort you have given me to make this thesis possible. I would also like to thank my lab mate Trenton Howell; he gave good insight and discussion along the way. Lastly, to my family and friends, new and old: thank you.

TABLE OF CONTENTS

Page

LIST OF TABLES

LIST OF FIGURES

LIST OF ABBREVIATIONS

Abbreviation	Description
CAVLC	Context-Adaptive Variable-Length Coding
FPGA	Field Programable Gate Array
ECC	Error Correcting Code
IDR	Instantanious Decoder Refresh
HVS	Human Visual System
LSB	Least Significant Bit
LUT	Look Up Table
MB	Macro Block
MBTM	Memory Bit Truncation Manager
MSB	Most Significant Bit
NAL	Network Abstraction Layer
RBSP	Raw Byte Sequence Payload
PSNR	Peak Signal to Noise Ratio
PSP	Picture Parameter Set
POC	Picture Order Count
ROI	Region of Interest
SCP	Secure Copy Protocol

SPS .... Sequence Parameter Set

SSIM .... Structual Simularity Index

IoT .... Internet of Things

VDD .... Supply Voltage

VR .... Virtual Reality

WE .... Write Enable

WPSNR .... Weighted Peak Signal to Noise Ratio

WNS .... Worst Negative Slack

YOLO .... You Only Look Once

ABSTRACT

Oswald, William, M. S., University of South Alabama, December 2021. Content-Adaptive ROI-Aware Video Storage Memory for Power Savings. Chair of Committee: Na, Gong, Ph.D.

The demand for mobile video streams is constantly increasing. With this demand comes a need for mobile devices to receive more videos at ever increasing quality. However, due to the large size of video data and intensive computational requirements, video streaming requires frequent memory access that consumes a substantial amount of mobile device power; as a result, the battery life of mobile devices is limited. In this thesis, a video content-adaptable Region-of-Interest (ROI)-aware video storage technique that promotes power savings is presented. During the video encoding process on the transmitting server, based on the macroblock variance and ROI characterization, the “macroblocks of interest” are identified and embedded in the encoded bitstream. In the decoding process, a new frame buffer with dynamic power-quality trade-off is presented to adapt to the macroblock characteristics during run-time. Results from the system-level and circuit-level simulations show that the proposed technique enables substantially more truncated bits and significant power savings while delivering similar or better video quality as compared to other state-of-the-art solutions.

CHAPTER I

INTRODUCTION

Mobile video streaming on YouTube, Vimeo, and Netflix has increased on average 70% per year and will consume approximately 79% of the total internet traffic by 2022 [1]. At the same time, power-efficient video storage has proven to be a very challenging problem to solve. This is due to the large data sizes associated and intensive computational requirements demanding frequent data access. With the advancement of computing technologies, more video streaming services deliver content to battery-powered mobile devices: such as smart phones and Internet-of-Things (IoT). On one hand, these devices would benefit greatly from low-power consumption as this would extend their battery life. On the other hand, the mobile video streaming process – receive, decode, and display of a video bitstream – consumes considerable power and limits the mobile devices' battery life. For example, with a video decoding chip, embedded memories contribute to over 50% of the decoding power consumption [2]. This use-case is only expected to grow for the next-generation video formats, H.265/HEVC and H.266/VVC, which has 2x-3x greater memory demands when compared to H.264 [3].

Today's mobile hardware designers, including memory designers, are focusing on hardware-level energy-efficient design techniques in order to accommodate the large amount of video data. However, these design techniques usually come with significant implementation overhead (e.g., silicon area, delay) to solve failure problems in memories. viewer-aware video memory was explored as a possible opportunity for power savings, taking advantage of the impact of illuminance levels in different viewing surroundings on the viewer's experience [4, 5, 6, 7], as shown in Fig. 1. Previous studies illustrate a new dimension of power savings for hardware design through the introduction of viewer awareness, but the developed memories lack runtime adaptation across a wide variety of mobile videos. To enable an optimized trade-off between power efficiency and video quality, this thesis aims to develop a video content-adaptable Region-of-Interest (ROI)-aware memory for general videos. Specifically, this thesis makes the following contributions:

- An intelligent ROI-aware and content-adaptive framework is proposed to determine video frame regions to preserve (output quality) or truncate bits for power savings. The truncation is applied for all Luma and Chroma video data (i.e., Y, U, and/or V components) (Chapter III & IV).
- The system-level implementation scheme of the proposed technique is developed and discussed (Chapter IV-A, IV-B, and IV-C).
- A low-power low-cost frame buffer with dynamic power-quality trade-off is developed to adapt to the video content (i.e., macroblock characteristics) during run-time (Chapter IV-D).
- A comprehensive suite of simulations on the proposed technique is performed and the enriched results are discussed, including the performance, circuit-level power efficiency, video-level power efficiency,

number of truncated bits, and output quality of various mobile videos (Chapter VI-A, VI-B, VI-C, and VI-D).

- An extensive statistical analysis demonstrates the effectiveness of the proposed technique in achieving significant bit truncations and power savings as compared to the state-of-the art, particularly for the videos with medium or high variance (Chapter VI-E).

With existing knowlege, this is the first work that seamlessly integrates ROI knowledge, i.e., “macroblocks of interest”, into the hardware design process.

The organization of the thesis is as follows: A review of low-power video memory designs is provided in Chapter II, Chapter III presents the macroblock variance and ROI study, Chapter IV discusses the proposed algorithm and software requirements. Chapter V shows the circuit and system level implimentation. A discussion of the evaluation methodology and results in Chapter VI and VII respectively. Chapter VIII compares the proposed tequnique against other alternative methods, and finally, a conclusion of the thesis is presented in chapter IX.[1]

[1] William Oswald was responsible for developing the software simulations, which included integrating macroblock variance information with ROI extraction. William was also responsible for the statistical analysis process, and video frame analysis. Dr. Ali Ahmad Haidous was in charge of the theoretical development, and the hardware system test platform. Hritom Das was in charge of all circuit design in Cadence, bit-level power analysis. Dr. Na Gong contributed to the theoretical development, and architecture.

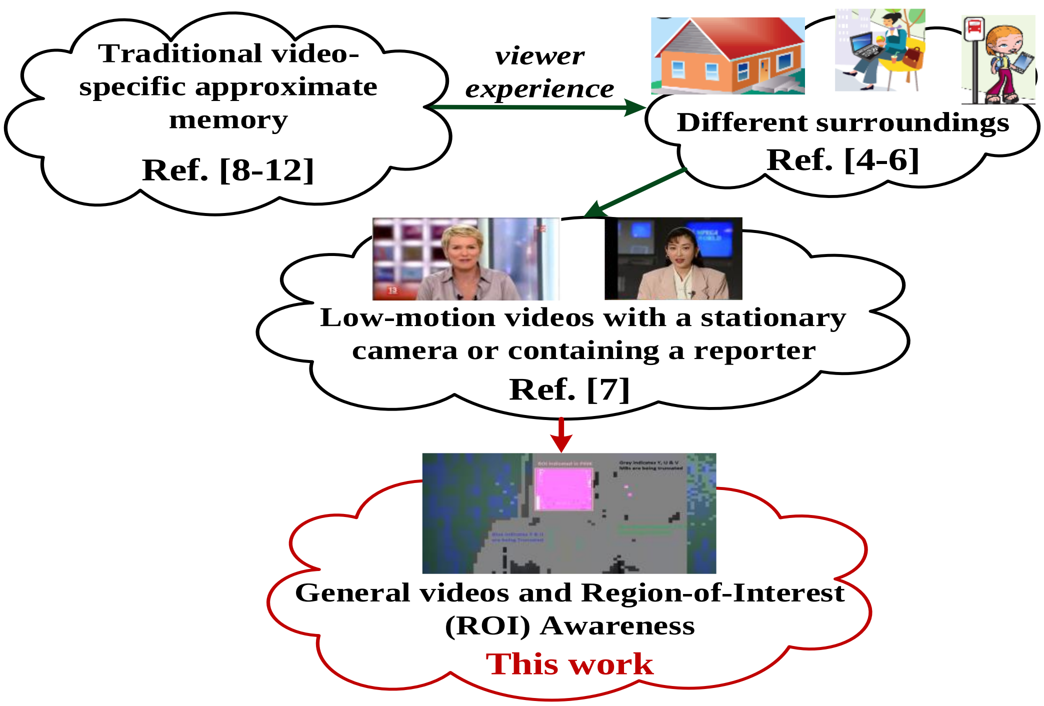


FIGURE 1 - Proposed content-adaptable ROI-aware low-power video memory.

CHAPTER II

STATE OF THE ART

A vast amount of research has been conducted to improve the power efficiency of video data storage. State-of-the-art, power-efficient video memories consist of either approximate memory with application-level information [8, 9, 10, 11, 12] or viewer-aware memories with an awareness of viewer's experience [4, 5, 6, 7]. In this Chapter, some of the existing work related to the proposed technique are briefly reviewed, and the detailed comparison analysis will be provided in Chapter VIII.

2.1 Approximate Video-Specific Memory

Researchers have presented various low-power video memory design techniques. Chang et al. [8] presented a hybrid 6T+8T SRAM to achieve quality-power optimization. Gong et al. [9] developed a hybrid 8T+10T memory for power savings based on the correlation between most-significant-bits (MSBs) of video data. In [10], a heterogeneous sizing scheme was presented to reduce the failure probability of conventional 6T bitcells. The video memory presented in [11] used the Least-Significant-Bits (LSBs) of video data to store the MSBs' error-correction-code (ECC). Kazimirsky et al. [12] developed a hybrid SRAM+DRAM memory to store MSBs in robust SRAM bitcells and LSBs in error-prone DRAM bitcells, leading to a tolerable output quality with power reduction.

However, all those video memory designs were developed without considering viewer's experience.

2.2 Viewer-Aware Video Memory

An investigation into viewer-aware low-power video memory techniques in was conducted, [4, 5, 6]: where an increased amount of ambient luminance allows for a larger number of bits to be truncated without noticeable degradation to the viewers. Very recently, the impact of video content characteristics on viewer's experience to enable video content-adaptive memory with dynamic energy-quality tradeoff was studed [7]. However, the technique determined the number of truncated LSBs based on the averaged plain macroblock percentage of an entire video sample; therefore, it was only effective to store low-motion videos with a stationary camera or containing a reporter in a video cast use-case. Additionally, this technique may result in noticeable distortion, e.g., a banding distortion caused by bit truncation, which negatively influenced the viewer's experience.

The common feature of these viewer-aware storage techniques is that the same number of the truncated bits were applied on an entire video. In contrast, the technique proposed in this thesis realizes content adaptation and ROI awareness within each video frame, thereby maximizing the number of truncated bits while maintaining the video quality.

CHAPTER III

OVERVIEW OF THE PROPOSED TECHIQUE

In this Chapter, the motivation of the proposed technique that introduces ROI awareness as bit truncation is applied for power savings is presented. Then, the high-level overview of the proposed technique is shown.

3.1 Motivational Example

Researchers conducted studies on the human visual system's (HVS) performance and concluded that viewers usually pay more attention to one or a few areas of a video and the region of concentration is called Region-Of-Interest (ROI) [13]. For example, in video conferencing applications, viewers typically pay more attention to the face regions than other areas. In video surveillance, the facial regions are what viewers concentrate most on in consecutive frames. Accordingly, ROIs have higher contribution towards the overall visual quality than other areas. Consequently, if truncation-caused banding distortion appears in ROIs, this will negatively influence a viewer's experience. Fig. 2 shows one example. The output quality of the video (Video tag: wF6lvdXXwc4 [14]) using the technique in [7] is shown in Fig. 2 (a). Since the banding distortion caused by bit truncation appears on the reporter's face, viewers were less likely to accept the displayed degradation due to this particularly noticeable distortion, as emphasized in [7].

Therefore, the motivation for this work arises from the following two observations:

1) In a video frame, the distortion in ROIs is more noticeable by viewers. Accordingly, if ROIs can be extracted and protected from truncation, the video quality would be improved from the viewer's perspective (Fig. 2 (b)). A comparison of the report's face using the technique in [7] and the proposed technique with ROI awareness is shown in Fig. 2 (c).

2) There existed a positive correlation between power savings and the number of bits truncated in a video decoder's frame buffer memory [7]. To optimize the power efficiency, it would be beneficial to increase the number of truncated bits in other regions which are not ROIs: the truncation regions.

(a) Output quality using [7] (at 3 truncated bits)

(b) Output quality of the proposed technique (at 3 truncated bits)

(c) [7] (left) vs. Proposed technique (right)

FIGURE 2. Observer discernable flaws in the facial region. This is due to a “banding effect” on the face when comparing (a) and (b) caused the overall quality of the frame to become unacceptable at 3 truncated bits (Video tag: wF6lvdXXwc4 from [14]).

3.2 Overview of the Proposed Context-Adaptable ROI-Aware Video Storage

Fig. 3 shows the proposed content-adaptable ROI-aware video storage technique. During the traditional mobile video streaming process, first, from (1) in Fig. 3, the mobile device requests a video for display from the cloud. Then, the streaming servers process the requested video by encoding and transmitting the encoded bitstream to the mobile device for decoding and display, (2) in Fig. 3. During this process, multiple memories are needed for storing the intermediate and final results of the frame data. In particular, the reference macroblock, frame memory, and display memory, which store the decoded video frames, are accessed very frequently, and they have a profound impact on the system's overall cost and power consumption. The proposed technique extracts ROIs in the cloud server and transmits the truncation region data together with the encoded bitstream to the mobile device, (3) in Fig. 3, to further reduce the mobile device's power consumption from computational overhead. The mobile device hardware video decoder receives the truncation region data and makes memory bit-truncation decisions for greater power savings with less perceived quality loss than [7]. To optimize the truncation decision logic of the mobile device hardware, which further improves power consumption, either *no truncation* or *3-bit truncation* is applied to the truncation regions. Explicitly, the proposed technique is detailed as follows.

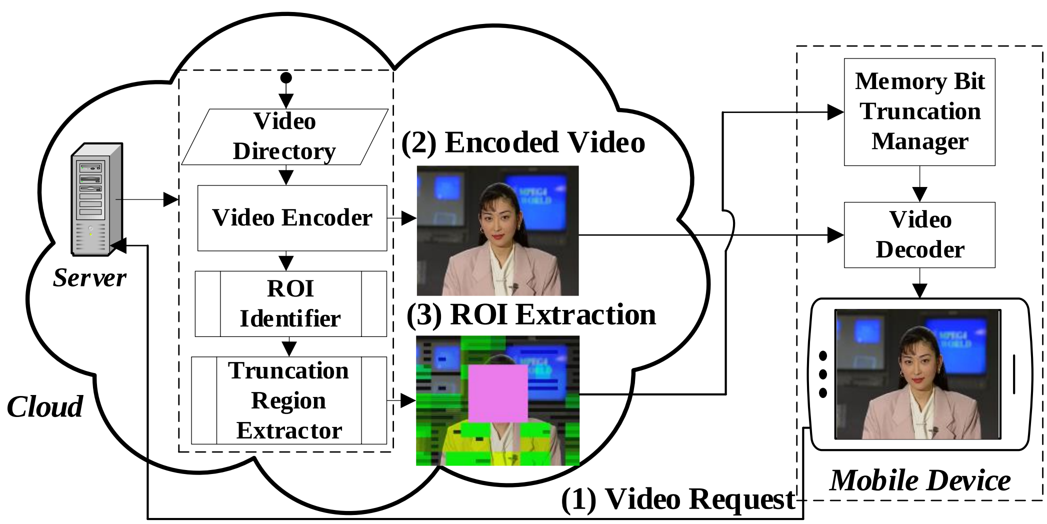


FIGURE 3. Proposed Region-Of-Interest and macroblock texture framework.

3.2.1 ROI Awareness

ROI has been recently applied for different research areas for video system optimization, such as wireless transmission [15], virtual reality (VR) [16], and video summarization [17]. The proposed technique introduces ROI awareness into video storage. Specifically, to minimize the complexity and computational overhead, the system focusses on the faces as ROIs in the analysis based on the basic machine learning facial detection OpenCV model [18]. Different algorithms, such as user attention model [13], motion-based models [17], and machine learning models [19], can be applied in our future investigations to extract different ROIs. It should be noted that the complexity of ROI extraction algorithms is also a trade-off choice between video quality and computation complexity as well as power savings. A simple ROI extraction algorithm will save computation resources and power consumption of video encoding. Also, it may transmit fewer truncation region bits to mobile devices, so more pixel bits will be truncated for power savings in the mobile devices. The drawback is that it will influence the video quality. Alternatively, a more complex ROI algorithm will identify additional regions and therefore it can convert a video without ROI to a video with ROI, which will benefit the video quality, but it will reduce the power savings due to the less truncated bits and increased computation complexity.

3.2.2 Video Content Adaptation

After the ROIs to preserve are detected and captured by the framework ROI Identifier, it then searches for regions of low variance measured by the percentage of plain macroblocks (MBs). Specifically, a MB defines an area of 16x16 pixels within a frame. An attribute associated with MBs is how "Textured or Plain" they are. A Plain MB is one in which the variance of intensity within the MB is less than or equal to the threshold value. It has been concluded in [7] that textured MBs are less susceptible to bit-truncation. To solve this, the pre-established method is usedfor determining the variance in a MB [20].

$$V_{MB} = \sum_{i=0}^{15} \sum_{j=0}^{15} (P(i,j) - \rho_{MB})^2 \gg 8 \qquad (1)$$

$$\mathrm{MB} = \begin{cases} \text{Plain}, & \text{if}(V_{MB} \le \mathrm{Th}_{low}) \\ \text{Textured}, & \text{Else} \end{cases} \qquad (2)$$

Equations (1) and (2), where ρ_{MB} is the average brightness within the MB, V_{MB} is the texture variance within the MB, and traditionally, Th_{low} is defined as a value of 1.25 [21].

3.2.3 Truncation Region Extractor

After ROIs are identified on the server, a truncation region extractor encodes the truncation region data using a proprietary protocol per frame and transmits in synchronization with the encoded video transmission to the mobile device. The truncation region data is decoded onboard the mobile device's hardware video decoder in a novel Memory Bit Truncation Manager (MBTM) hardware unit: which truncates a novel frame buffer memory through the use of unique control YUV truncation signals. The video decoding and bit truncation processes occur in lockstep.

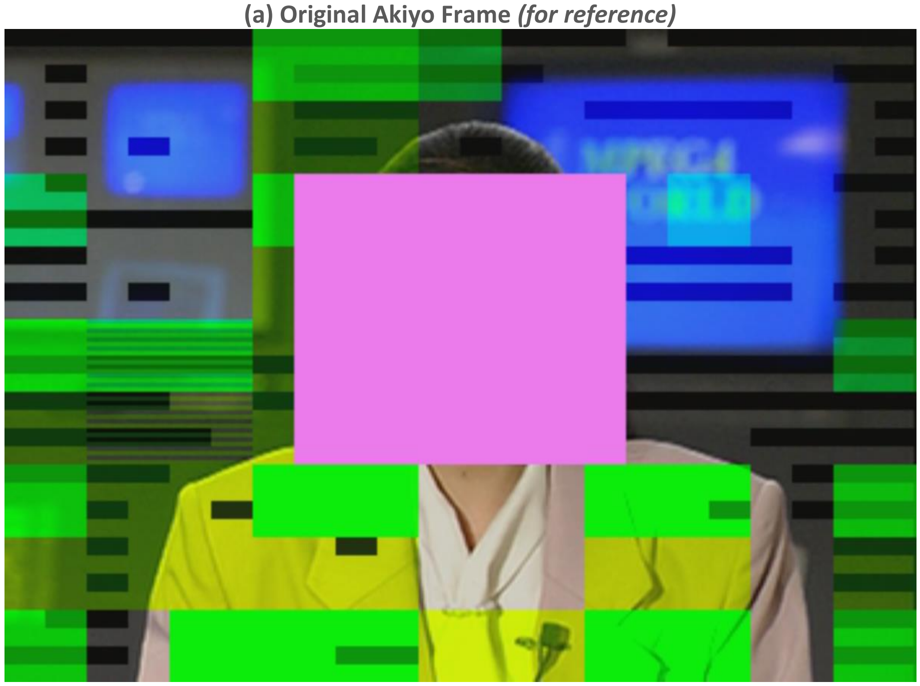
3.2.4 3-Bit Truncation

Truncation is performed in the YUV (Y'CbCr) color space [22], inferring that any truncation is done to the YUV color values. The memory designed in [7] truncated 1, 2, or 3 bits in the Least Significant Bits (LSBs) of the Y vector 2 of all frames within an entire video as a blanket truncation. The proposed technique will enable a different number of truncated bits for each region within each frame within an entire video. To minimize the implementation overhead, only 3-bit truncation is adopted in the new frame buffer, which will be discussed in Chapter V-D. Meanwhile, the proposed technique can identify bit-truncation for each Y, U, and V vector of the frame separately for each truncation region in each frame, instead of only truncating the Y vector as a blanket truncation across the entire video as the existing techniques [4, 5, 7]. Furthermore, the proposed technique is expected to enable additional bit truncations as compared to existing techniques. Also, to minimize the video quality degradation caused by bit truncation, the developed frame buffer truncates three LSBs to the optimal value "100" [7], instead of truncating the values to "000".

Fig. 4 shows the *Akiyo* video sample using the proposed technique. The extracted preserved ROI region is highlighted in pink. All truncation regions within a frame are identified, including the following seven possible truncation combinations: (1) Green, Y vector truncation; (2) Blue, U vector truncation; (3) Yellow, V vector truncation; (4) Dark blue, YU vectors truncation; (5) Dark Yellow, UV vector truncation; (6) Dark green, YV vectors truncation; and (7) Grey, YUV vectors truncation. Each of these combinations would be encoded in the truncation region data for the MBTM to generate control signals for memory bit truncation in the video decoding process.

To conclude, our proposed technique truncates the chroma sub samples within each frame as well as the luminosity: Y, U, and V vectors. Previous research only targeted luminosity, Y, of a video for truncation, while chroma samples were disregarded for the entire video. Also, our technique preserves ROIs that impact viewer perception most, while enabling greater truncation for each Y, U, and V vector for the truncation regions with textured MBs. Accordingly, the proposed technique will realize a greater number of truncation while preserving visual quality. The system-level and circuit-level implementations of the proposed technique will be discussed in Chapter V.

(a) Original Akiyo Frame *(for reference)*



(b) Visualized ROI Sample

FIGURE 4. Akiyo frame visualization [23]. Generated using proposed method's frame parsing process. Pink, preserved ROI. Seven possible truncation combinations: 1. Green, Y vector truncation. 2. Blue, U vector truncation. 3. Yellow, V vector truncation. 4. Dark blue, YU vectors truncation. 5. Dark Yellow, UV vector truncation. 6. Dark green, YV vectors truncation. 7. Grey, YUV vectors truncation.

CHAPTER IV

ROI EXTRACTION AND TRUNCATION ALGORITHM

In this chapter, the methodologies behind the ROI extraction algorithm will be explained in detail, as well as the choice to use a standard public solution instead of devleoping a custom ROI algorithm for this circuits needs.

4.1 ROI Algorithm Selection process

The choice in selecting an ROI extraction algorithm is heavily dependent on the type of video being displayed. For instance, the user’s attention will change drastically if they ware watching a ports game, as compared to a news broudcast. For this reason, standard object detection algorithms do not fully satisfy the requirement of generalizing ROI locations within any video stream. Subsequently the development of a general solution at determining ROI within any video stream would be a novelty, and is outside the scope of this thesis. With this constraint in mind, two factors went into deciding what ROI extraction algorithm to use.

1. An assumption will be made in determining what the user’s attention will be in a video stream. This allows a single region extracting algorithm to be equivalent to a ROI extractor.

2. The region extractor used should be publicly available, and highly reputable such that the effectiveness of the ROI is not a concern.

This led to the decision to use OpenCV open-source repository of various frontal face detection algorithms [18]. With this repository in mind, the decision was made to target news broadcasting video streams as the sample target video genre. Using the assumption that the user's attention will be located on the faces within these video streams.

4.2 Haar Cascade Classifier for ROI Extraction

Within the OpenCV repository, all the available facial detection algorithms were tested to find an optimal algorithm, of which 'HaarCascade_FrontalFace_mlt2.xml' was selected, as it provided the easiest interface, and was a very predictable and algorithmic approach to facial detection. However, it should be noted that any facial detection algorithm could have been used, such as the Eigen vector approaches, Fisher's Linear Discrimination Analyzer, or Local Binary Pattern apprioch [23]. The model chosen uses the Haar cascade approach for facial recognition. This apporch uses digital image processing to translate an input image into a feature set, then from this featureset a classifying machine learning algorithm is used to distinguish if the image contains a face or not. This classifier was trained using the 'Open Images V4' dataset, which contains 15.4 million bounding box images, which was used to train the classifier [24]. This implies that the classifer follows the 'You Only Look Once' (YOLO) methodology, and thus no time information is considered in detecting a face within an image. Thus, all frames within a video are treated indepentently from oneanother.

4.3 Memory Structure and YUV Colorspace

The face detection classifier used for ROI extraction naturally does not consider the underlying memory structure of the decoder when classifying images. For this reason, any region defined as a ROI from the classifer needs to be mapped to a specific byte in memory before truncation. In this specific usecase, the target decoder is the H.264 decoder, which uses a 16x16 pixel macroblock within the memory structure. To ensure that all the ROI is preserved in the output of the decoder, a conservative decision was made, such that if a single pixel within a macroblock is defined as an ROI, the whole macroblock is preserved. It is important to note that the H.264 decoder uses the YUV colorspace in the memory structure [2]. Thus, any bits truncated in memory effect the YUV colorspace, and does not directly affect the RGB output of the decoder. The YUV colorspace is used throughout the decoder, and is translated to the RGB colorspace in a conversion circuit before the video stream leaves the decoder. Thus, the effect of truncating a single YUV color byte will influence a 16x16 pixel region on the RGB display.

4.4 Complete Truncation Algorithm

With the ROI extraction classifier in place, and macroblock variance defined with equation (2), it is possible to process any frame. The algorithm Implimented works as such:

1. Find ROI within frame via ROI extraction algorithm, store ROI macroblock locations in memory
2. Calculate Macroblock Variance via Equation (2) for each macroblock.
3. If a macroblock is defined as non-ROI from step #1, and High Variance from Step #2, truncate the macroblock memory cell.

This algorithm is very effective in that steps #1 and #2 can be done in parralell using entirely different CPU cores, or different integrated circuit structures. This would theoredically allow such an algorithm to be effective even for live video broadcasting.

CHAPTER V

PROPOSED TECHNIQUE: SYSTEM AND CIRCUIT LEVEL IMPLEMENTATION

This Chapter presents the system-level and circuit-level implementation of the proposed technique.

5.1 System Level Implementation: Video Streaming Platform

Fig. 5 shows the developed system-level video streaming platform. As shown, a Raspberry Pi [25] microcontroller was used to serve as a video streaming server with which a mobile device would communicate and retrieve video data. Also, a Z-Turn 7020 [26] board and synthesized an H.264 video decoder was utalized into the on-board Xilinx Zynq 7020 Field Programmable Gate Array (FPGA) which would operate as a mobile device. Finally, the decoded video data was captured via a Magewell [27] HDMI Video Capture & Display Device.

The corresponding block diagram for Fig. 5 is illustrated in Fig. 6. The video streaming process is kicked-off by a command from the mobile device to the server to retrieve an encoded H.264 video stream over Secure Copy Protocol (SCP) [28]. The mobile device sends the initial kick-off command to the server over a serial terminal on a PC interfaced with the mobile device over USB. The server then processes the video stream requested by the mobile device by both transmitting an H.264 encoded format of the video stream over SCP to the mobile device and parsing the frames for truncation region information.

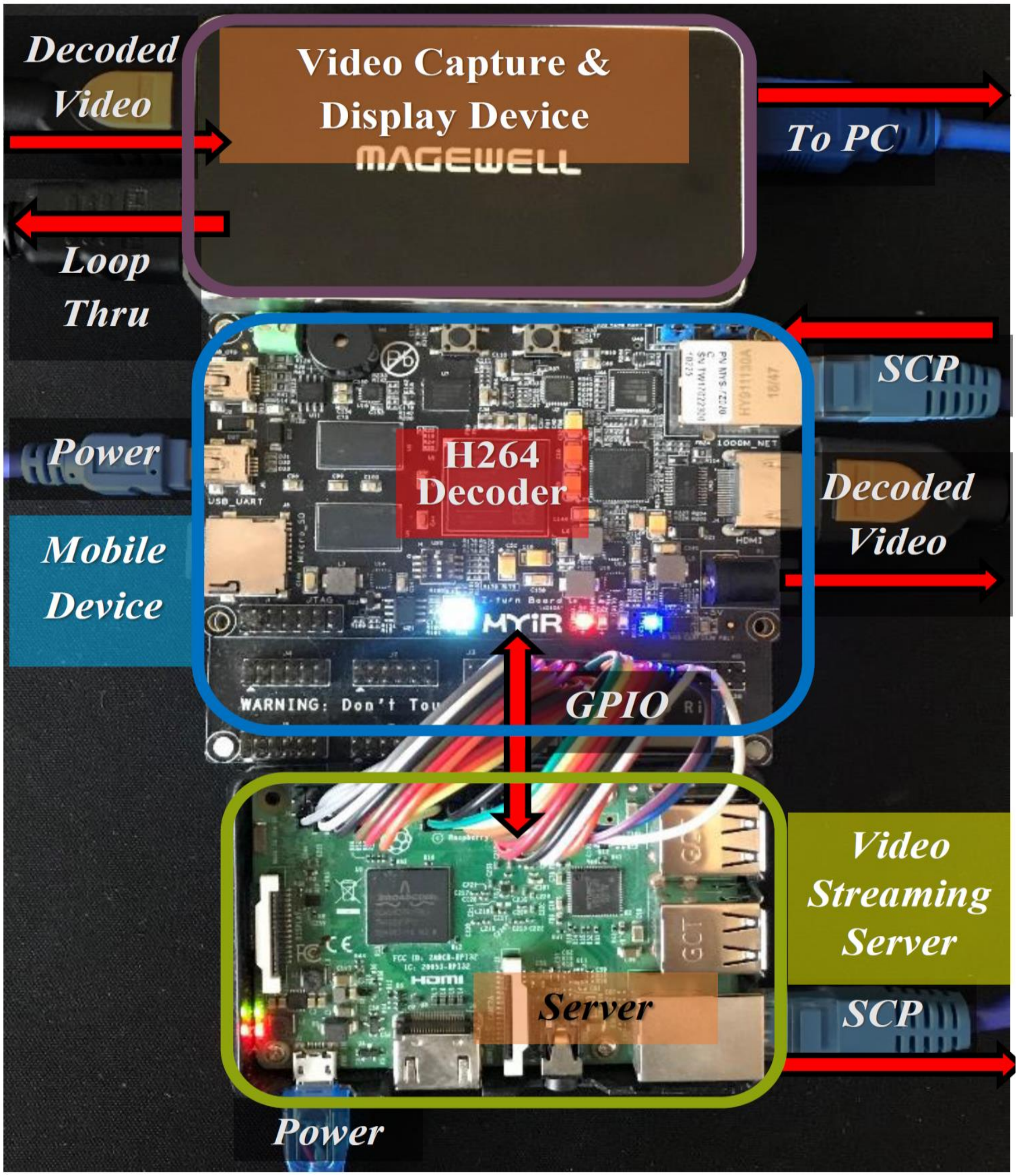


FIGURE 5. H264 video stream demonstration platform hardware system.

TABLE 1. Truncation Region GPIO Protocol.

(a) SERVER-TO-MOBILE DEVICE

Index 0	Index 1	Index 2	Index 3	Index 4	…	Index N+1	Index N+2	Index N+3
Frame Number	Number of Regions	YUV_1 Truncation	$(X_1 1,Y_1 1)$	$(X_1 2,Y_1 2)$	…	YUV_N Truncation	$(X_N 1,Y_N 1)$	$(X_N 2,Y_N 2)$
22 bits	16 bits	3 bits	22 bits	22 bits	…	3 bits	22 bits	22 bits

(b) MOBILE DEVICE-TO-SERVER

Index 0	Index 1
Frame Number Request	Send Frame Flag
22 bits	1 bit

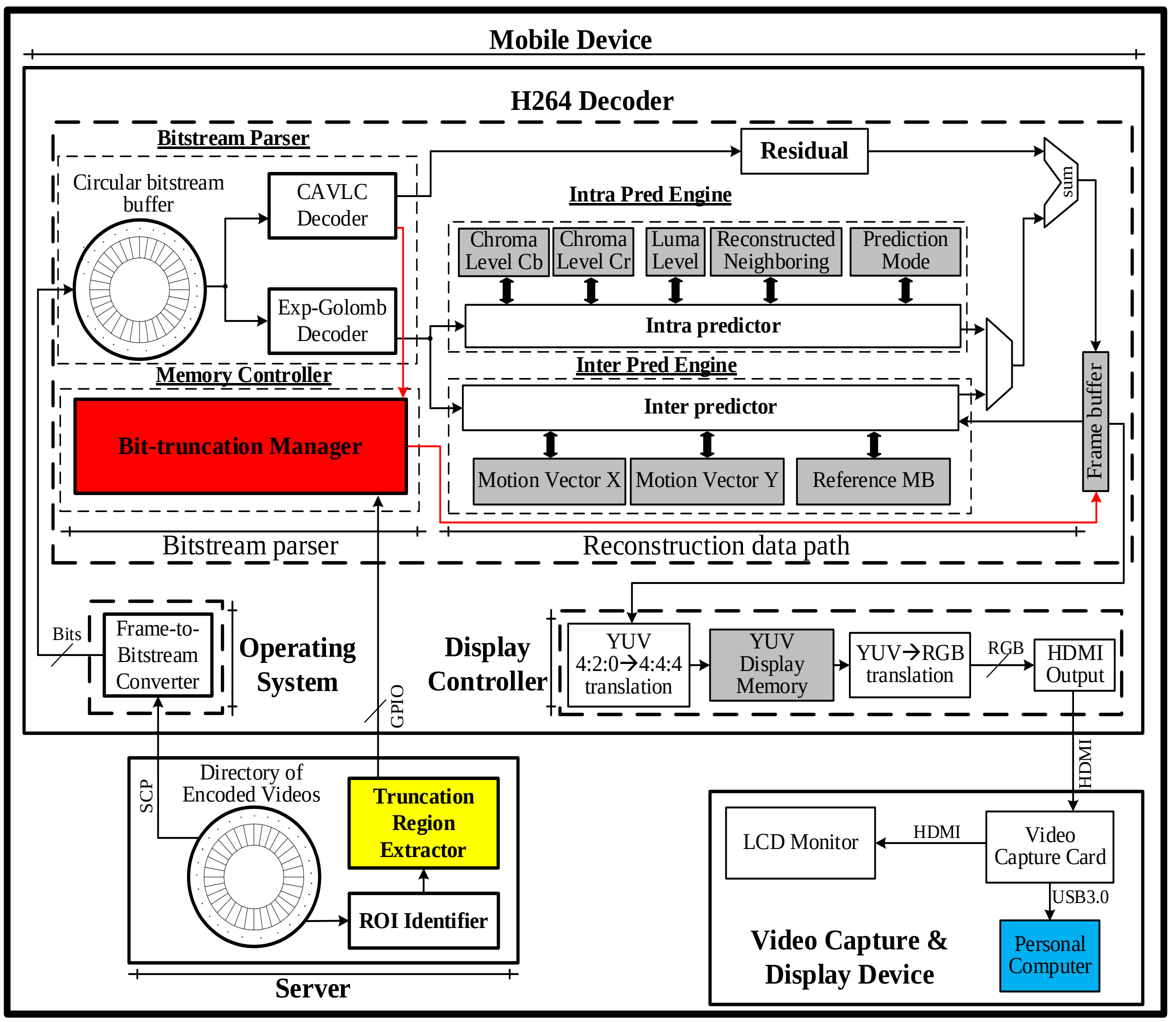


FIGURE 6. Mobile video steaming system block diagram.

After the video frame is parsed on the server, the truncation region information is transmitted over GPIO per frame. In our developed system, the protocol is defined inTable 1. Only the truncation region information of the frames that would be truncated is transmitted. The preserved ROI information will not be transmitted as these regions are identified prior to the transmission on the server and preserved. As listed in Table 1, the first index, index 0, denotes the current frame number parsed. The second index, index 1, denotes the number of truncation regions to truncate. Then the next indices denote the first three YUV truncation signal bits plus two sets of XY coordinates denoting the left top and right bottom corners of rectangles grouping the affected truncation region. These three indices repeat for each region called out by the “Number of Regions”, index 1. The GPIO interface data width bit size of the developed system is 22-bits per index. The 22-bit distribution is to account for a maximum of 211 x 211 pixel addressing – a max resolution of 1,920 × 1,080 – totaling 22 bits. There is an additional 2 handshaking bits between the server and mobile device to denote data reception confirmation in-order to transmit the next index.

This truncation region information will be transmitted to a MBTM for processing in the mobile device side, as discussed in Chapter V-B. The MBTM will generate control signals for the frame buffer memory, thereby determining which sub-pixels – from Y, U, and/or V – shall be truncated for each frame written to the frame buffer memory, which will be detailed in Chapter V-D. Finally, the decoded and bit-truncated frame is output over HDMI from the mobile Device and captured by the Video Capture & Display Device.

5.2 Memory Bit Truncation Manager (MBTM)

The MBTM implemented into the H.264 decoder parses the protocol data that is transmitted by the server's Truncation Region Extractor. The flow is broken down as follows. First, from Fig. 7 (a), the encoded frame is transmitted via SCP to the mobile device. Fig. 7 (b) illustrates the truncation regions determined to be bit-truncation capable on a sub-frame vector level: Y vector, U vector, and V vector each encompassing all the sub-frames summing to a frame. From Fig. 7 (b), the gray areas denote the truncation regions determined to be bit-truncation capable for all Y, U, and V vectors. The areas in boxes are regions where only 1 or 2 vectors were determined to be bit-truncation capable. Two coordinates, top-left and bottom-right, are highlighted in Fig. 7 (b) for each of these regions to show how the truncation region data was used to determine the regions to truncate using the protocol in Table 1. A total of 61 regions to truncate are shown in Fig. 7 (b). Fig. 7 (c) shows the resultant frame after Fig. 7 (a) is decoded using the identified truncation region information. As shown, the preserved ROI around the face, pink region from (b), is not truncated to avoid visual quality degradation. The frame is decoded normally, but when it is written into the frame buffer, the transmitted truncation region information is used to control the T_Y, T_U, and T_V control inputs to truncate the frame buffer memory as it is written. These control inputs are provided to the proposed frame buffer in Fig. 8, which will be discussed in detail in Chapter V-D.

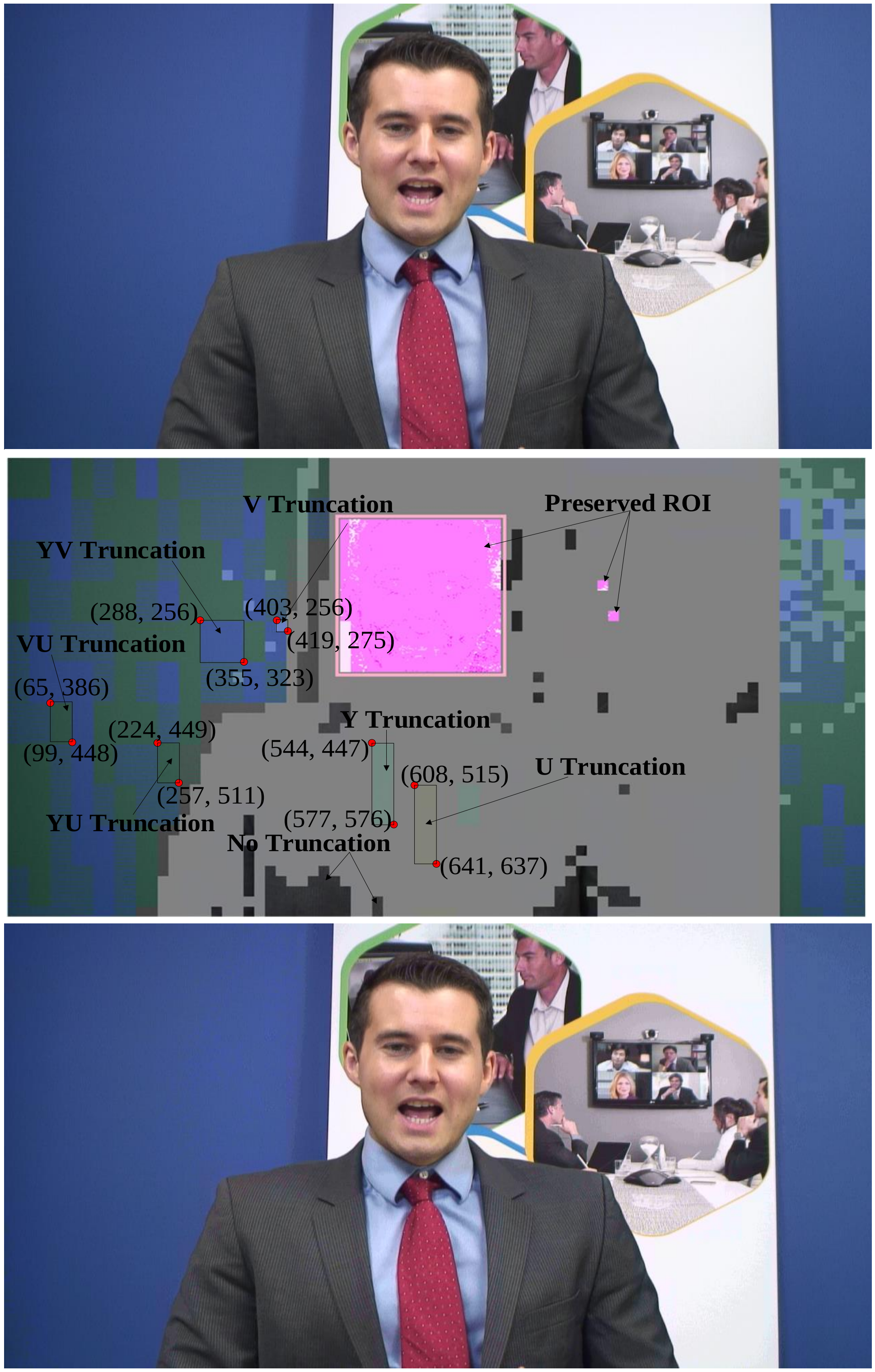


FIGURE 7. (a) Encoded frame 175 from Johnny_1280x720_60 video [23]. (b) Visual of areas being truncated. 45 regions total. (c) Output decoded frame. 2,282,496 bits truncated.

5.3 H.264 Decoder and MBTM Integration

A H.264 video decoder is implemented based on the Open Source Osenlogic OSD10 decoder IP [29]. This decoder was capable of decoding baseline profile level 3.1 encoded bitstreams. The slice types supported were I-Slice, SI-Slice, P-Slice, and SP-Slice [30]. The entropy coding profile supported was Context-Adaptive Variable-Length Coding (CAVLC). The decoder took an H264 Network Abstraction Layer (NAL) bitstream and output YUV 4:2:0.

During the NAL bitstream parsing process, the bitstream is parsed into raw bytes of syntax elements from the Raw Byte Sequence Payload (RBSP). Within the RBSP, therein lies the slice layers. Ignoring the Sequence Parameter Set (SPS) and the Picture Parameter Set (PPS), the Instantaneous Decoder Refresh Access Unit (IDR Slice(s)) and the slice layer includes all slice headers and slice data for the frames that shall be truncated using the MBTM. H.264/AVC defines a frame as an array of luma samples and two corresponding arrays of chroma samples: denoted as YUV.

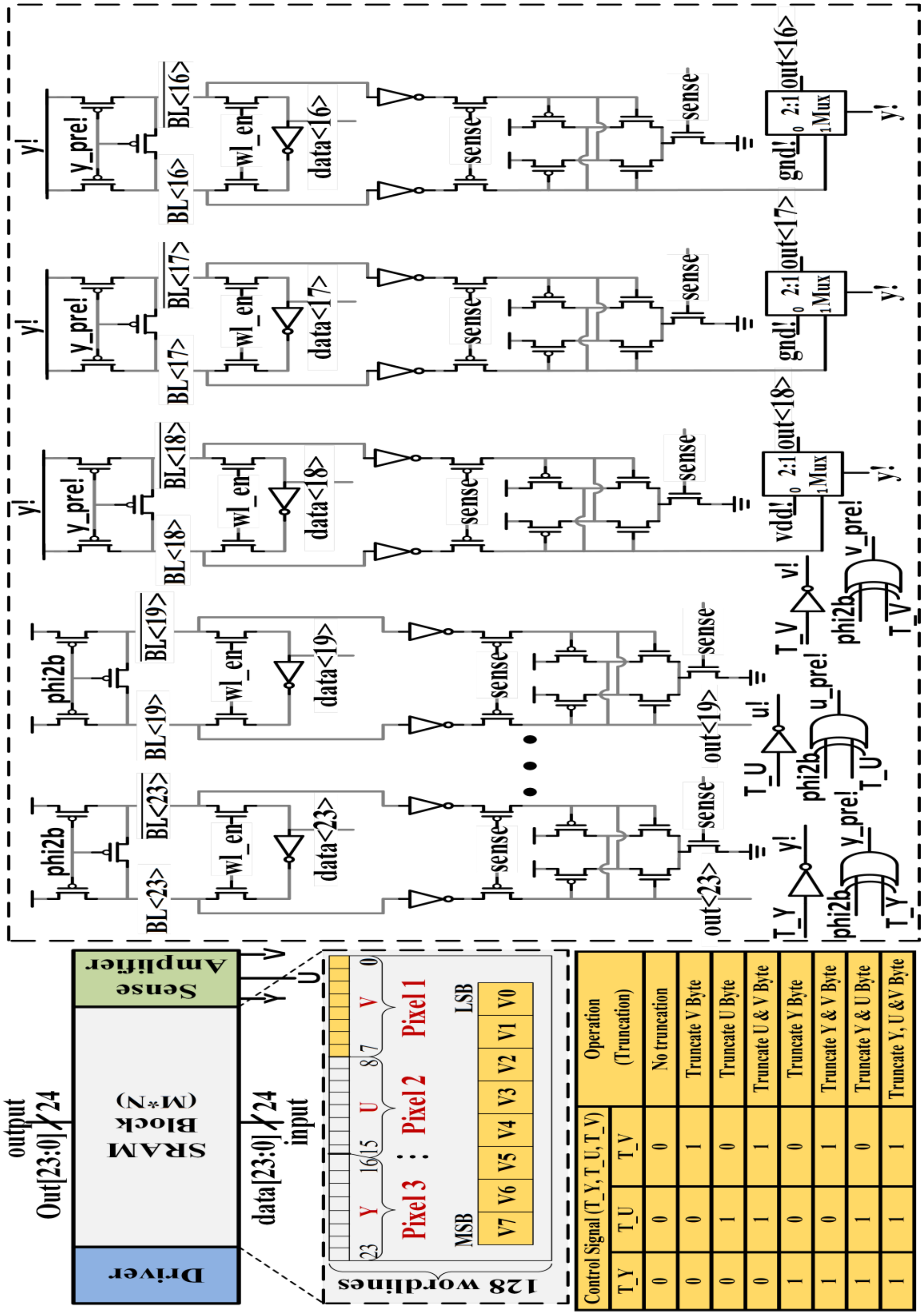
Specifically, the slice header includes the parameters *first_mb_in_slice*, which indicates the position of the first macroblock in the slice data, and *frame_num*, which represents the order in which a video decoder shall decode the encoded frames. This is not the same as the display order or Picture Order Count (POC), which is the order in which the frames are displayed. The *frame_num* parameter is used to determine which frames during the decoding process would be susceptible to YUV bit-truncation by the MBTM and the *first_mb_in_slice* is used to determine the starting coordinates of the macroblocks susceptible to bit-truncation. The slice data included all the macroblocks of the slice.

After the MBTM determined that a frame would be truncated, through a conditional match between the frame number parameter from Table 1(a) and *frame_num*, a running count of the current macroblock index was kept track of internally to the MBTM from the slice data starting with the index of *first_mb_in_slice*. After the MBTM determined that a macroblock would be truncated, through a conditional match of the running macroblock index and the truncation region given by the two indices from Table 1 (a) that indicate the rectangular region which YUV truncation would be applied, the MBTM passes through the YUV truncation signal, from Table 1 (a), to the frame buffer which would result in the macroblock being truncated to the desired amount. An internal signal denoting the number of macroblocks truncated in the frame is then incremented. After all the macroblocks desired to be truncated in the frame are truncated, denoted by the number of ROI parameter from Table 1 (a), then from Table 1 (b), the Send Frame Flag is set then reset by the MBTM over GPIO to signal the next frame information to truncate. From Table 1 (b), the Frame Number Request index is used to fetch any frame index truncation information for macroblocks that required multiple frames for prediction. This process is repeated until the end of the NAL bitstream.

The trade-off with utilizing the BTM is the additional GPIO parallel bitstream overhead required to truncate the macroblocks in each frame. Each frame parsed had an absolute worst case overhead of approximately 380,738 additional bits to transmit using the protocol from Table 1. This worst case is calculated assuming every macroblock with 16×16 pixels in a maximum resolution of $1,920 \times 1,080$ would be truncated differently per frame in a video. On average, however, the number of additional bits transmitted per frame is 1,200, because the maximum resolution of each frame is 1920×1080 and the truncation regions are combined to encompass a greater area in the video to save on bits transmitted: on average 50 truncation regions per frame. With a 1920 × 1080 video at 30 frames per second progressive (1080p 30fps) or a 1280 × 720 videos at 60 frames per second progressive (720p 60fps), i.e. 5,000 kbps bit rate, the worst case percentage overhead would be 7.62% with an average of 0.02% per frame. The protocol utilized is one of the simplest methods to implement the proposed technique.

5.4 Circuit Level Implementation of the Proposed Frame Buffer Memory

During the video decoding process, multiple memories are needed. In particular, the frame buffer memory is accessed very frequently and it has a profound influence on the system's overall cost and power consumption [7]. In this thesis, a new frame buffer is designed, and the circuit-level implementation is shown in Fig. 8. Specifically, the logic in the truth table highlighted in yellow was designed to be supported by the MBTM. Here, T_Y, T_U, and T_V are utilized to truncate Y, U, and V byte from the word. Each word consists of a Y, U, and V byte. During the Write Enable (WE) phase of the frame buffer memory access, if either control line of T_Y, T_U, and / or T_V are asserted, the memory would truncate the 3-LSB of the optimal asserted vector as "100" [7]. The proposed frame buffer has M words and each word consists of N bits. To evaluate the functionality and measure average power consumption of this proposed circuitry, a 128-word by 24-bits memory array is designed. Here, input and output pins are denoted as data[23:0] and out[23:0] respectively. Bits 23-16 are named Y byte, bits 15-8 are named



Control Signal (T_Y, T_U, T_V)			Operation (Truncation)
T_Y	T_U	T_V	
0	0	0	No truncation
0	0	1	Truncate V Byte
0	1	0	Truncate U Byte
0	1	1	Truncate U & V Byte
1	0	0	Truncate Y Byte
1	0	1	Truncate Y & V Byte
1	1	0	Truncate Y & U Byte
1	1	1	Truncate Y, U &V Byte

FIGURE 8 Circuit-level implementation of the proposed frame buffer memory

U byte, and bits 7-0 are named V byte. The memory implemented had a driver U byte, and bits 7-0 are named V byte. The memory implemented had a driver and sense amplifier for writing and reading data. These enabled bits truncation according to T_Y, T_U, and T_V control signal activation. If T_Y, T_ U, and T_V are all de-asserted as logic ‘0’, then the frame buffer would operate as a traditional memory device where the sense amplifier would operate with a supply voltage (VDD) and pre-charge signal phi2b. When the T_Y signal is asserted as logic ‘1’, the peripheral circuitry would generate two signals: y! which is the inverted value of T_Y and y_pre! which is inverted value of the pre-charge enable signal. These two signals are used to control the sense amplifier for the Y byte’s 3-LSBs, thereby enabling truncation. During this process, the VDD for this sense amplifier remains grounded and the pre-charge signal would be reactivated. As a result, the power consumption of this portion of circuitry will be reduced as compared to the normal operation. During the read back operation, the 3-LSBs are generated as “100” though use of three 2:1 multiplexers in-place of regular of data output. When the bit truncation is asserted, these multiplexers would select “100” through control signals y!, u!, or v!. Otherwise, these multiplexers would pass normal readout data values. In addition, the VDD of all the 3 LSBs of each byte are also controlled by the corresponding control signals y!, u!, and v!. During the truncation, VDD for LSBs can be powered off to save power consumption and multiplexers will select “100” as the output data, thereby achieving low-power operation. The detailed timing diagram and power efficiency of the proposed memory will be discussed in Chapter VII.

CHAPTER VI

EXPERIMENTAL METHODOLOGIES

This Chapter discusses the metrics, methods, and strategies used to evaluate the effectiveness of the proposed technique. The testing and analysis setup used to generate the experimental results is also discussed.

6.1 Video Selection

To verify the effectiveness of the proposed technique, 74 videos with diverse characteristics were selected from the YouTube 8M dataset [14], YouTube UGC dataset [31], and Xiph.org Video Test Media [32]. As shown in Fig. 13, those videos have different resolutions (e.g. 288 × 352, 1280 × 720, and 1920 × 1080) and different MB variance characteristics (low, medium, and high). Of those videos, 60 videos contain facial features to enable ROI preservation using the proposed technique. All videos were converted to the YUV 4:2:0 chroma subsampling standard for ease of bit-truncation. A detailed statistical analysis shows that our selected videos are representative of the full population of videos in general, which will be discussed in Chapter VII-E.

6.2 Video Frame Quality Metrics

Existing video quality metrics such as PSNR and structural similarity (SSIM) [33], which are used widely to evaluate the video quality, but it fails to incorporate the importance of ROI. This is because these metrics weigh all pixels of the video equally, regardless of user awareness. For this reason, another video quality metric – Weighted Peak Signal to Noise Ratio (WPSNR) – is used in this thesis to evaluate the quality of videos with ROI [22], which is defined as [34]:

$$\mathrm{WPSNR} = 10log_{10}(255^2 / D_{frame}) \tag{3}$$

$$D_{frame} = \boldsymbol{\alpha} * \mathrm{MSE}(\mathrm{f}, \mathrm{f}') + (1 - \boldsymbol{\alpha}) * \mathrm{MSE}(\mathrm{f}, \mathrm{f}') \tag{4}$$

Where MSE stands for the Mean Squared Error between the original frame and after truncation while $\boldsymbol{\alpha}$ (alpha) is defined as the weight that the ROI would have. The $\boldsymbol{\alpha}$ value will be a constant value of 0.9 following the previous research in [22]. This combines PSNR with ROI information, however such an ROI weighted metric is not widely accepted for SSIM. For this reason, videos with ROI information will be evaluated using WPSNR, whereas videos without ROI information will have both PSNR and SSIM.

6.3 System and Circuit Level Implementation

The hardware system platform from Fig. Fig. 5 implemented an H264 decoder synthesized into a Xilinx Zynq XC7Z010 FPGA fabric. The H264 decoder IP Core was designed using the Xilinx Vivado 2019.2 [35] software design suite. This same decoder is modified to include an MBTM. The FPGA was commanded via an ARM Cortex-A9 Processor running on a Linux Operating System through a custom baseband driver.

The circuit-level frame buffer is implemented using a 45nm CMOS technology [36]. The supply voltage is 1.0V. The memory size is 128 words at 24 bits per word.

6.4 Video Quality Evaluation

All selected videos were analyzed using an in-house custom software tool. The tool operated in the following three-step process: (i) Load one original video frame from memory; (ii) Apply both the method in [7] and the proposed method to the original frame and generate the truncated frame using each method; and (iii) Compare the frames generated against the original frame and calculate the PSNR, SSIM and WPSNR values. With data points collected on a per-frame basis, the average PSNR, SSIM and WPSNR of each video stream was calculated and compared.

6.5 Statistical Hypothesis Validation

From the proposed method, a hypothesis was conjectured: that the differences between the method in [7] and the proposed method follow a Normal, or near-Normal distribution. This should hold true for both PSNR and WPSNR. To support this hypothesis, a goodness of fit regression test was preformed to determine if the data falls within the probability plot of a Normal or Weibull distribution. If the data follows this hypothesis, this would suggest that the sample set of videos is of adequate size and as a result, no more videos would need to be tested.

CHAPTER VII

EXPERIMENTAL RESULTS

7.1 Mobile System Utilizing Proposed Method Overhead

Fig. 10 and Fig. 11 show the post-implementation project summaries of the baseline H264 decoder and the H264 decoder modified to include an MBTM. When comparing both figures, one observes that the Lookup Table (LUT) overhead, which is the additional logic gates required for the proposed design over the baseline, was 204 LUTs or a 0.38% increase in area. The I/O, which was used for the server-to-mobile device interface, increased by 37, or 29.6%. The power consumption of the modified decoder also increased by 0.068 watts or 0.03%: most of which was attributed to the increased number of I/O.

TABLE 2. Summary of SYSTEM Overhead/Cost Using the Proposed Method at 1920 × 1080 Resolution

	Description	**Data**
Bitrate: Server to Mobile Device	Additional bits transmitted from server to mobile device for protocol per frame	1,200 bits or a 0.02% increase on average per frame
Power Consumed: Mobile Device	Additional power consumed on the mobile device in Watts	0.068W or 0.03% increase due to additional I/O
Network Overhead	Additional time needed for additional protocol data to transmit per frame over the 4G LTE network	Between 240µs and 100µs more time per frame on 4G LTE at 5Mbps and 12Mbps
Logic Gates: Mobile Device	Additional Look Up Tables (LUTs) needed on the FPGA implementing the Proposed Method on the mobile device	204 LUTs or a 0.38% increase in area

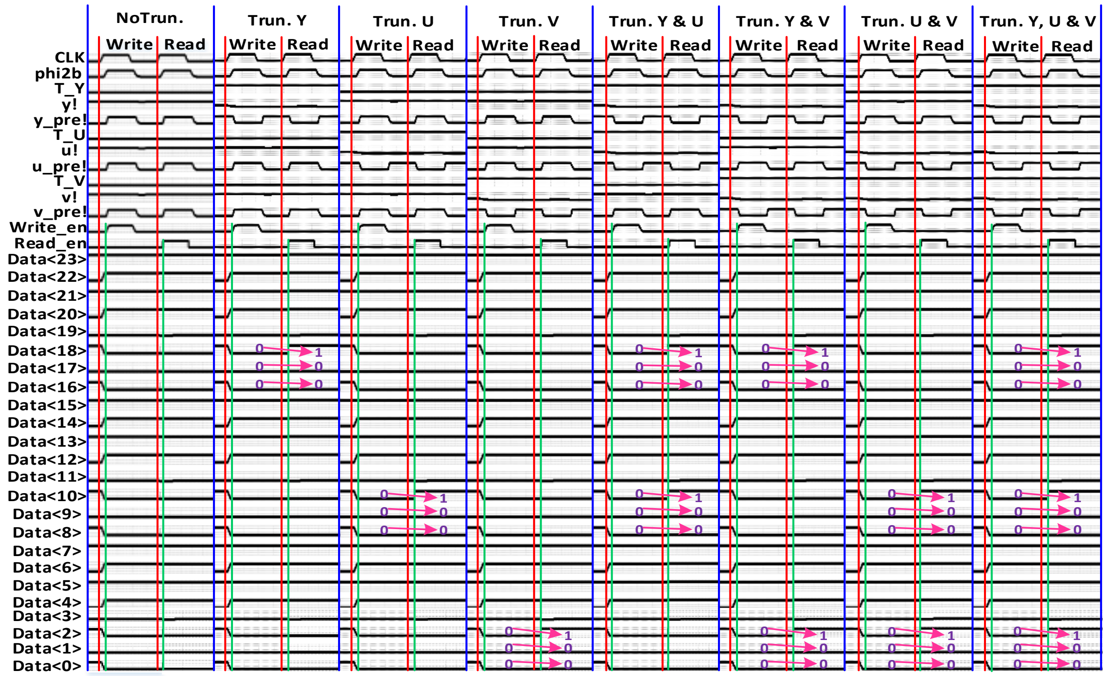


FIGURE 8. Timing diagram of the frame buffer circuit.

Finally, the Worst Negative Slack (WNS) increased by 0.011ns, which was within acceptable tolerance for this system as any positive value means that the critical path passes timing constraints. Overall, this additional overhead was tolerable when compared against the benefits in power savings and quality improvements achieved using the proposed technique.

Table II presents the summary of all the overhead associated with the system implementing the proposed method with video resolution1920 × 1080, which is the maximum resolution supported by the system. The primary advantage of the proposed method is the power savings achieved onboard the mobile device's H264 hardware decoder frame buffer memory, discussed later from Fig. 12. The disadvantages are the bitrate, power consumption, network, and logic gate overhead. From Table 2, the mobile device needs to receive 1,200 additional bits on average per frame from the server. This coupled with 204 additional LUTs required and 240µs of additional network uptime result in a 0.03% increase in mobile system power consumption.

7.2 Circuit Level Frame Buffer Timing Diagram

The proposed frame buffer is shown in Fig. 8 and the simulation timing diagram is shown in Fig. 9. In this waveform, phi2b, T_Y, y!, and y_pre! denote the pre-charge (for

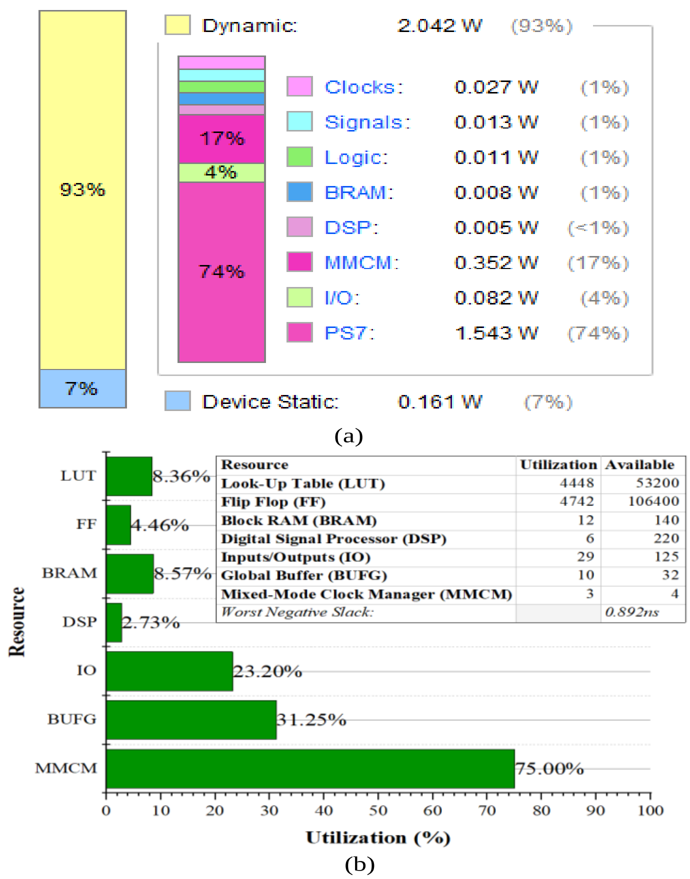


FIGURE 9. Hardware FPGA system post- implementation project summary *without* BTM. (a) On-Chip Power, Total Power: 2.203W. (b) Resource allocation.

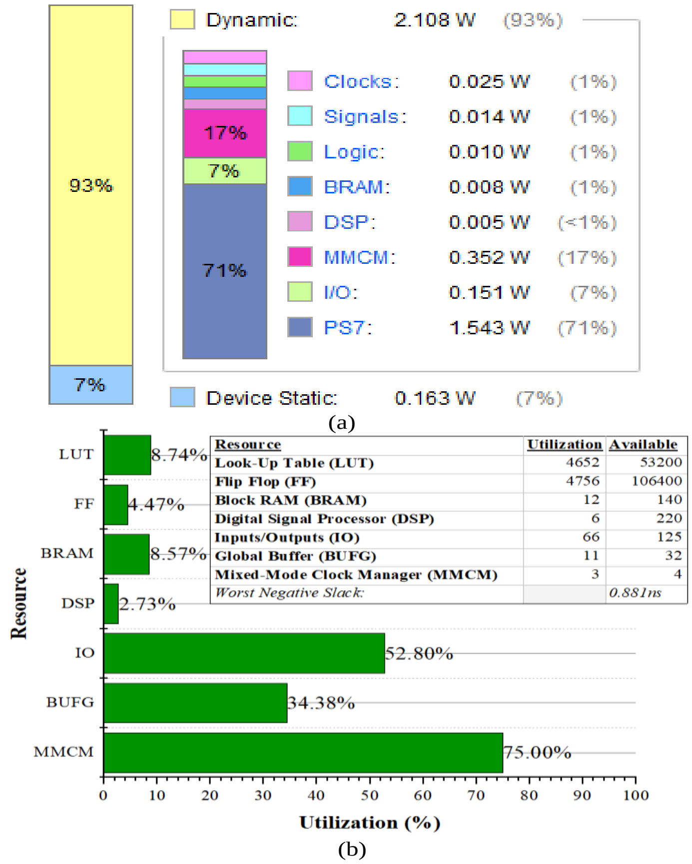


FIGURE 10. Hardware FPGA system post-implementation project summary *with* BTM. (a) On-Chip Power, Total Power: 2.271W. (b) Resource allocation.

un-truncated bits), bit truncation enable for Y byte, power supply for truncated bitcell's (last 3 LSBs of Y byte), and pre-charge deactivated signal for truncated bit cells, respectively. T_U and T_V controlled the bit truncation for U and V bytes respectively. Write and read enable signals initiated the write and read operations for the memory accordingly. Data [23:0] were the three bytes of each word of the proposed memory buffer. Here, blue to red lines stand for "don't care" regions. The red lines denote where the rising clock edge was initiated for write and read operations. Finally, the green lines denote that the write and read operations were enabled. All 8 truncation permutations and traditional read and write operations were presented in the timing diagram as an exhaustive simulation of the frame buffer circuit.

It should be noted that, if the bit truncations were initiated, then 3 LSBs were truncated from the selected byte/bytes based on the control signals T_Y, T_U, and T_V. During the read operations, the 3-LSBs of the truncated bytes would output "100" bits through the utilization of 2:1 multiplexers instead of being read from memory to save power.

7.3 Circuit Level Frame Buffer Power Savings Analysis

Fig. 12 presents the power consumption of the proposed frame buffer in all eight possible conditions, including seven truncation cases and one baseline case without bit truncation. Specifically, the eight cases include: (i) No truncation with control signals T_Y & T_U & T_V ='0', (ii) Y vector truncation with T_Y='1', (iii) U vector truncation with T_U='1', (iv) V vector truncation with T_V='1', (v) Y and U vectors truncation with T_Y & T_U ='1', (vi) U and V vectors truncation with T_U & T_V='1', (vii) V and Y vectors truncation with T_V & T_Y ='1', and (viii) YUV vectors truncation with T_Y & T_U & T_V ='1'. As discussed in Chapter IV-B, for the truncated vectors, the three LSB will be truncated to "100" to maximize power savings. All 8 truncation cases presented in Fig. 12 are tested in 6 ways: when written ('0' to '0', '0' to '1', '1' to '0', '1' to '1') and when read back ('0' & '1'). The power consumed in each case was calculated, and then the average is presented.

At first, a random word was initialized with (A5A5A5)16, then the same memory word was immediately read back with (F0F0F0)16, then all the ‘1’s and ‘0’s written and read received the same priority in the power consumption calculations. The same word consumed 3.90E-4 W power without any bit truncation. When the circuitry selected any T_Y, T_U or T_V control option, where 3-LSBs were truncated from each one selected, 6.67% power was saved when compared against no bit truncation. When T_Y & T_U, T_Y & T_V or T_U & T_V were selected, where 3-LSBs were truncated from each selected byte, then 13.33% power was saved when compared against no bit truncation. Finally, when T_Y, T_U, and T_V were selected for truncation, where 3-LSBs were truncated from each selected byte, then 19.74% power was saved. The supply voltage for this simulation was 1V, where the proposed frame buffer circuit can operate to specification and had no faulty bit(s).

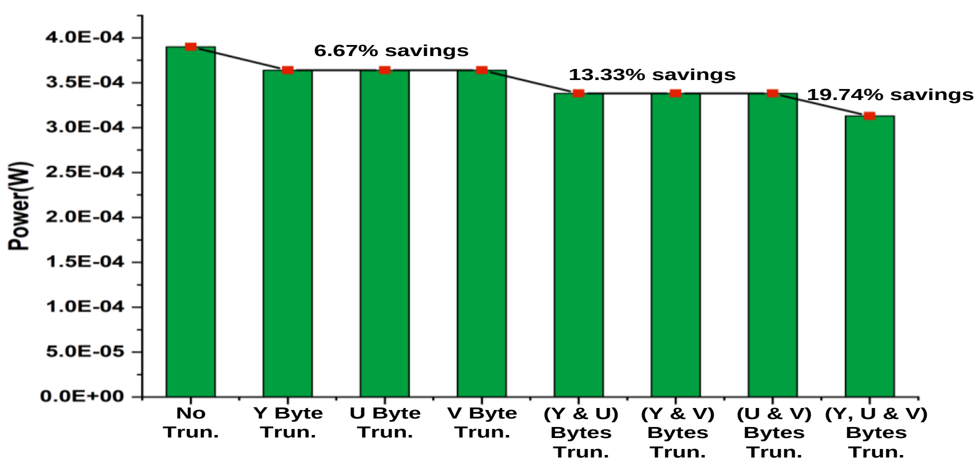


FIGURE 11. Power savings (one word) of the frame buffer circuit.

7.4 Video Visual Quality Comparisons

Fig. 13 shows visual frame comparisons for three selected videos with ROI between the proposed method and [7]. The proposed technique enables significant visual quality improvement as compared to [7]. Specifically, for the Foreman_cif video, due to the truncated LSBs in [7], the man's cheeks, forehead, and hat shadows experience noticeable banding distortion, negatively affecting video quality. Alternatively, the proposed ROI-aware technique effectively reduces the banding distortion and improves the visual quality. Similarly, with [7], the mother_daughter_cif demonstrates banding distortion around the cheeks and hair, and the carphone_qcif video suffers from discoloration around the cheeks and chin. The introduced ROI awareness of the proposed technique effectively avoids losing the quality of videos. Another observation from Fig 13 is that the proposed technique achieves a much higher WPSNR value of all three videos. A more detailed analysis on WPSNR will be provided in the next sub-chapter.

7.5 Objective Video Quality and Bit Truncation Analysis

Fig. 13 compares WPSNR values and the number of truncated bits of 60 videos with ROI using the proposed technique to the state-of-the art [7]. As shown, the proposed technique can enable 26.46% additional truncated bits as compared to [7]. Meanwhile, with the ROI awareness, the proposed technique can effectively enhance the quality of the majority of videos. On average, the proposed technique can increase the WPSNR values by 20.17% videos, as compared to [7].

Further analyziong the impact of the MB variance characteristics (low, medium, and high variance) on the effectiveness of the proposed technique. The results are shown in Fig. 14. As can be seen, the WPSNR improvement strongly depends on the MB variance of videos. Specifically, videos with high variance achieve the most significant quality improvement using the proposed technique, with 47.31% WPSNR increase on average. With the proposed technique, all videos with medium variance also demonstrate quality improvement, with 13.74% WPSNR increase on average. However, the proposed technique shows little video quality improvement for videos with low variance and even results in minimal video quality degradation (with 1.75% WPSNR loss on average. This suggests that the proposed technique is particularly effective for videos with high and medium MB variance.

Analyzeing the results of 14 videos without ROI. As shown in Table 3, the proposed technique can enable a significant number of truncated bits, with a minimal PSNR drop. On average, 44.61% additional truncated bits can be achieved, with 3dB PSNR loss.

Finally, the average SSIM was calculated for all video streams without ROI to verify the quality drop within each video. Table 4 verify that the quality loss within each video was minimal. With the average loss in each video being 0.0223 for videos with ROI, and 0.0167 for videos without ROI respectively. This is the expected result, as videos without ROI experience color truncation in all regions of the video, thus the user is more susceptible to noticing quality loss.

7.6 Video Level Power Savings Analysis

To compare the power effectiveness of the proposed ROI-aware technique to the traditional memory design and the state-of-the art [7], a model was built to simulate the power consumption of the memory for a video as:

$$\begin{cases} P(Video_i) = \dfrac{1}{N_i}\sum_{j=1}^{N_i} P_k(j) \\ k \in (0,1,2,3) \end{cases} \qquad (3)$$

where Ni is the total number of bytes for the video i, $Pk(j)$ is the normalized power consumption to store byte j with k truncated bits. For the proposed memory, $k = 3$; for the traditional memory, $k = 0$; for the memory in [7], $k = 0$, 1, 2, or 3. For a fair comparison, the normalized power consumption $Pk(j)$ is based on the power consumption reported in [9]. The results are listed in Fig. 13 and Table 3. As observed, the proposed technique only consumes 83.79% and 76.56% total power on average for videos with ROI and videos without ROI, respectively, as compared to the traditional memory. Also, the proposed technique achieves 3.06% and 8.26% power savings for videos with ROI and videos without ROI, respectively, as compared to [7]. It is worth mentioning that, our analysis only considers the facial features as ROI of videos and integrating advanced ROI identification algorithms will covert videos without ROI to videos with ROI, thereby further increasing the effectiveness of our proposed technique to general videos.

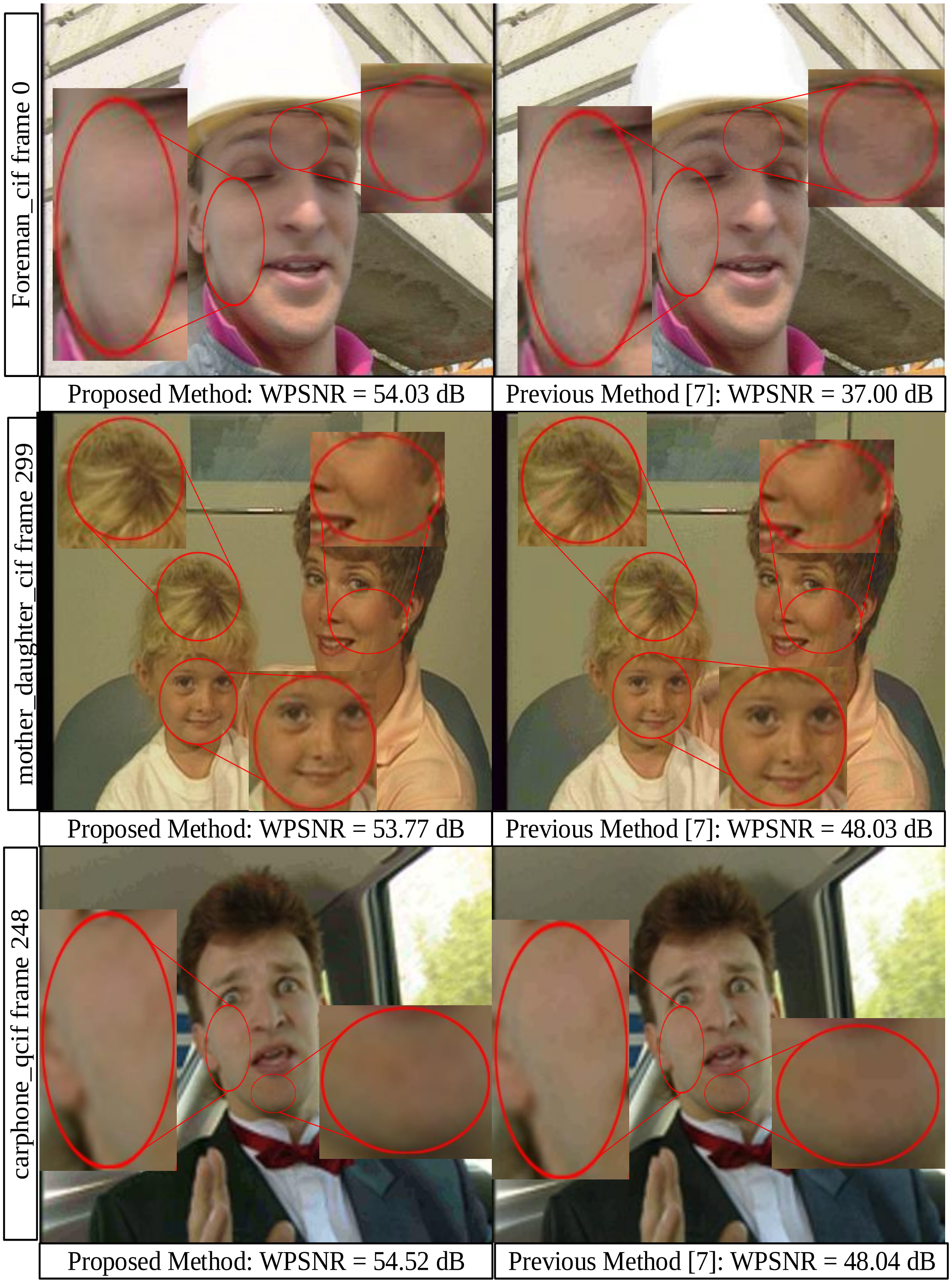


Figure 13. VISUAL COMPARISON OF SELECTED VIDEO FRAMES

7.7 Statistical Analysis

Various videos were analyzed using the proposed method. In-order to confirm that the selected video analysis results are a representation of the full population of all videos, a statistical analysis of the results was conducted. to verified the statistical analysis to determine that the results are relevant across all videos *not* analyzed. Specifically, the Pearson's Chi-square test [37], which is also known as the Chi-Squared *goodness-of-fit* test, is used in our analysis. The goodness-of-fit test checks whether the sample data is likely to be from a specific theoretical distribution, and therefore represents the data expected in the actual population. The idea is, if the sample data does fit an expected distribution, then it shows that the sample data represents the full population of the video data in existence. The statistical results will either reject or accept the working statement called the *null hypothesis*, H0, which is the opposite of the *alternative hypothesis,* H1. To reject or accept the null hypothesis, several methods exist, one of which is the Probability value method i.e. *P-Value method*. The *P-Value* is the evidence against the null hypothesis, i.e., the smaller the P-Value, the stronger the evidence that the null hypothesis should be rejected. The P-Value method is based on a critical value, which is determined based on the distribution. For example, if the data shows a *normally distributed* population – which according to the statistical results shown later, this critical value is a *z-score*. The z-score is a value that is then used to lookup the P-Value in a Standard Normal *z-table,* which is used to then test the null hypothesis. If a P-Value is greater than an alpha or **α** value of 0.10, then the statistical results are "not significant" and thus, the null hypothesis is accepted. However, if the P-Value is less than or equal to **α** values of 0.05 or 0.01, then the results are “significant” or “highly significant” respectively, and thus, the null hypothesis is rejected in favor of the alternative hypothesis. The rejection regions depend on the confidence level that the results are significant, e.g., if a confidence level is 95%, then an α value of 5% or 0.05 is chosen: 100% - 95%.

In our analysis, the null hypothesis for the Chi-Squared goodness-of-fit test, H0, is, “For the given set of video data points, a specified distribution accurately represents the data”, and therefore, the alternative hypothesis, H1, is, “For the given set of video data points, a specified distribution *does not* accurately represent the data.” Hence, the goal of the statistical analysis is to validate the null hypothesis and thus deduce that the specified distribution would fit the data. To achieve this statistical result, P-values were calculated for each data set – low, medium, and high variance – for WPSNR metrics, Power Savings, and video noise introduced. The Chi-Squared goodness-of-fit test can only be used for data put into classes (or bins); therefore, the data sets are put into histograms: Figs 14, 15, 16, and 17 used the MathWave Technologies EasyFit software[38], to find the Chi-Squared goodness-of-fit test, in order to determine the type of distribution. In the video analysis results, the WPSNR metrics, power savings, and video noise were calculated and introduced for all 74 videos for both the truncation method in [9] and the proposed method. As well, the data was split into three sets referred to as low, medium, and high variance, which corresponded to 1-bit, 2-bit, and 3-bit truncation videos using the truncation method in [9], respectively. These data are what are refer to as video data points in our statistical analysis.

Fig. 15 demonstrates how categorizing the data creates clear groupings when comparing the truncation method in [7] to the proposed method. The figure shows three distinct 3-parameter Weibull distributions that describe the quality improvement between the proposed and [7]. These Weibull distributions are within the 95% confidence interval required. Each distribution reports a P-value greater than 0.1, implying that we cannot reject the null hypothesis and accept this distribution as a possible representation of the data. Fig. 16 shows the power savings distribution for each video type as a 3-parameter Weibull distribution. Power savings is reported as a percentage increase, using the total number of bits truncated in each video and the power consumption shown in Fig. 12. All of these distributions pass the 95%

confidence interval. 17 shows the probability of noise increase in a random video stream. All distributions shown fall into the category of normal distributions with a 95% confidence interval. Fig. 18 shows the probability of quality drop measured in SSIM for a random video stream, with a 95% confidence interval that the probability lies within a normal distirbution. The most notable differences between the Figure 16 and Figure 17 is that the Medium and Low Variance videos show very little difference for the SSIM loss, whereas the loss is very distinct in the PSNR distribtuion.

It was determined that because all videos are compared to themselves for improvement, e.g., video after the proposed method is applied verses the original video, video resolution has no statistical impact in the data set. Power Consumption will be presented by improvement percentage, thus ignoring linear growth in watts saved in larger scale videos. Similarly, it is statistically sound that a larger dataset is not needed to affirm the distributions. As all distributions shown fall within the 95% confidence interval, there is only a 5% chance that the data collected is far from the specified distribution.

In summary, videos categorized as high variance show the biggest improvements in WPSNR quality, the most power saving by percentage, and introduce the least noise as measured by PSNR and SSIM. With medium variance videos also saving on power consumption, with a more noticeable drop in quality and increase in noise. As such, videos classified as low variance often have little to gain using this method, and sometimes even cause video quality degradation.

TABLE 3. Results of *ROI* Videos

Videos with ROI	Truncated bits			Normalized power consumption			WPSNR (Alpha = 0.9)		
	Ref. [7]	Proposed	**diff**	Ref. [7]	Proposed	**diff**	Ref. [7]	Proposed	**diff**
akiyo_cif	30,412,800	32,922,081	**8.25%**	90.97%	93.55%	**-2.83%**	57.53	55.14	**-4.16%**
claire_qcif	50,079,744	44,009,266	**-12.12%**	90.97%	94.76%	**-4.16%**	57.61	57.10	**-0.88%**
dinner_1080p30	1,969,920,000	1,629,113,461	**-17.30%**	90.97%	95.07%	**-4.50%**	57.32	58.21	**1.56%**
grandma_qcif	88,197,120	78,023,685	**-11.53%**	90.97%	94.73%	**-4.12%**	57.57	57.21	**-0.62%**
intros_422_cif	36,495,360	43,295,433	**18.63%**	90.97%	92.93%	**-2.15%**	57.45	55.15	**-4.01%**
Johnny_1280x720_60	553,881,600	456,595,131	**-17.56%**	90.97%	95.09%	**-4.52%**	57.07	58.44	**2.39%**
KristenAndSara_1280x720_60	553,881,600	488,596,289	**-11.79%**	90.97%	94.74%	**-4.14%**	56.99	57.37	**0.65%**
miss_am_qcif	15,206,400	10,966,364	**-27.88%**	90.97%	95.70%	**-5.20%**	57.60	58.54	**1.63%**
news_cif	30,412,800	36,162,420	**18.91%**	90.97%	92.91%	**-2.13%**	57.52	54.48	**-5.27%**
rush_hour_1080p25	1,036,800,000	1,134,324,798	**9.41%**	90.97%	93.48%	**-2.75%**	57.49	55.78	**-2.97%**
sign_irene_cif	54,743,040	64,431,060	**17.70%**	90.97%	92.98%	**-2.21%**	57.48	55.01	**-4.29%**
trevor_qcif	15,206,400	16,565,578	**8.94%**	90.97%	93.50%	**-2.78%**	57.31	54.99	**-4.05%**
vidyo1_720p_60fps	553,881,600	597,801,678	**7.93%**	90.97%	93.57%	**-2.85%**	57.54	56.05	**-2.58%**
west_wind_easy_1080p	1,181,952,000	1,086,498,282	**-8.08%**	90.97%	94.52%	**-3.90%**	56.81	55.68	**-1.99%**
720p50_mobcal_ter	928,972,800	1,325,076,458	**42.64%**	86.60%	82.99%	**4.17%**	47.88	54.16	**13.12%**
720p50_shields_ter	928,972,800	1,245,591,004	**34.08%**	86.60%	84.01%	**2.99%**	47.69	54.48	**14.25%**
aspen_1080p	2,363,904,000	3,041,639,112	**28.67%**	86.60%	84.66%	**2.24%**	47.92	54.97	**14.71%**
blue_sky_1080p25	899,942,400	1,060,438,048	**17.83%**	86.60%	85.95%	**0.75%**	47.66	55.10	**15.61%**
bowing_cif	60,825,600	76,915,166	**26.45%**	86.60%	84.92%	**1.94%**	48.02	53.69	**11.81%**
bridge_close_cif	405,504,000	560,890,106	**38.32%**	86.60%	83.51%	**3.57%**	47.95	54.13	**12.88%**
carphone_qcif	77,451,264	88,390,034	**14.12%**	86.60%	86.39%	**0.24%**	48.04	54.52	**13.48%**
controlled_burn_1080p	2,363,904,000	2,937,098,762	**24.25%**	86.60%	85.18%	**1.63%**	47.96	55.18	**15.06%**
crew_4cif	121,651,200	172,691,140	**41.96%**	86.60%	83.07%	**4.07%**	47.54	53.54	**12.61%**
crowd_run_1080p50	2,073,600,000	3,005,452,078	**44.94%**	86.60%	82.72%	**4.48%**	47.42	53.55	**12.93%**
deadline_cif	278,581,248	334,134,316	**19.94%**	86.60%	85.70%	**1.04%**	48.02	54.48	**13.45%**
FourPeople_1280x720_60	1,107,763,200	1,318,759,148	**19.05%**	86.60%	85.80%	**0.92%**	47.96	55.12	**14.94%**
Lecture_1080P-412e	1,034,726,400	998,909,402	**-3.46%**	86.60%	88.49%	**-2.18%**	48.07	57.01	**16.91%**
life_1080p30	3,421,440,000	4,187,455,252	**22.39%**	86.60%	85.41%	**1.38%**	48.04	54.99	**14.46%**
mother_daughter_cif	60,825,600	79,283,536	**30.35%**	86.60%	84.46%	**2.47%**	48.03	53.78	**11.95%**
pamphlet_cif	60,825,600	83,561,818	**37.38%**	86.60%	83.62%	**3.44%**	47.93	53.27	**11.14%**
paris_cif	215,930,880	302,557,572	**40.12%**	86.60%	83.29%	**3.82%**	47.88	53.81	**12.40%**
pedestrian_area_1080p25	1,555,200,000	1,749,179,384	**12.47%**	86.60%	86.59%	**0.01%**	48.05	55.59	**15.70%**
rush_field_cuts_1080p	2,363,904,000	3,080,806,912	**30.33%**	86.60%	84.46%	**2.47%**	47.86	54.09	**13.01%**
salesman_qcif	91,035,648	127,208,586	**39.73%**	86.60%	83.34%	**3.77%**	47.97	53.74	**12.03%**
station2_1080p25	1,298,073,600	1,803,902,208	**38.97%**	86.60%	83.43%	**3.66%**	47.79	53.71	**12.39%**
students_cif	204,171,264	283,317,790	**38.76%**	86.60%	83.45%	**3.63%**	47.92	53.64	**11.93%**
sunflower_1080p25	2,073,600,000	2,874,084,316	**38.60%**	86.60%	83.47%	**3.61%**	47.81	53.79	**12.52%**
suzie_qcif	30,412,800	31,039,198	**2.06%**	86.60%	87.83%	**-1.42%**	48.07	55.79	**16.07%**
touchdown_pass_1080p	2,363,904,000	2,715,649,830	**14.88%**	86.60%	86.30%	**0.35%**	47.94	55.38	**15.52%**
tractor_1080p25	2,861,568,000	4,031,161,306	**40.87%**	86.60%	83.20%	**3.92%**	47.50	53.67	**12.99%**
vidyo3_720p_60fps	1,107,763,200	1,368,042,822	**23.50%**	86.60%	85.27%	**1.53%**	48.11	54.54	**13.36%**
vidyo4_720p_60fps	1,107,763,200	1,206,225,090	**8.89%**	86.60%	87.02%	**-0.48%**	48.13	55.78	**15.89%**
720p50_parkrun_ter	1,393,459,200	2,074,332,336	**48.86%**	82.11%	73.37%	**10.64%**	36.71	53.47	**45.63%**
720p5994_stockholm_ter	1,669,939,200	2,434,415,832	**45.78%**	82.11%	73.93%	**9.97%**	35.49	53.38	**50.41%**
ducks_take_off_1080p50	3,110,400,000	4,661,102,586	**49.86%**	82.11%	73.20%	**10.86%**	36.36	53.30	**46.61%**
football_422_cif	109,486,080	161,366,445	**47.39%**	82.11%	73.64%	**10.32%**	36.54	53.96	**47.67%**
football_cif	79,073,280	114,873,738	**45.28%**	82.11%	74.02%	**9.86%**	36.58	53.79	**47.03%**
foreman_cif	91,238,400	123,714,882	**35.60%**	82.11%	75.75%	**7.75%**	37.01	54.04	**46.01%**
hall_monitor_cif	91,238,400	136,539,606	**49.65%**	82.11%	73.23%	**10.82%**	36.66	53.56	**46.12%**
harbour_4cif	182,476,800	268,811,991	**47.31%**	82.11%	73.65%	**10.31%**	36.34	53.91	**48.34%**
ice_4cif	145,981,440	183,650,610	**25.80%**	82.11%	77.50%	**5.62%**	35.22	52.91	**50.21%**
mobile_calendar_422_cif	109,486,080	163,476,849	**49.31%**	82.11%	73.29%	**10.74%**	36.37	53.06	**45.88%**
old_town_cross_420_720p50	1,382,400,000	2,060,977,467	**49.09%**	82.11%	73.33%	**10.69%**	36.44	53.60	**47.07%**
riverbed_1080p25	1,555,200,000	2,285,697,270	**46.97%**	82.11%	73.71%	**10.23%**	36.62	53.29	**45.54%**
silent_cif	91,238,400	130,669,137	**43.22%**	82.11%	74.38%	**9.41%**	36.70	53.46	**45.64%**
soccer_4cif	182,476,800	249,118,389	**36.52%**	82.11%	75.58%	**7.96%**	36.49	53.36	**46.25%**
tennis_sif	45,619,200	67,173,204	**47.25%**	82.11%	73.66%	**10.29%**	36.27	54.29	**49.69%**
tt_sif	34,062,336	50,343,861	**47.80%**	82.11%	73.56%	**10.41%**	36.16	54.24	**49.98%**
vtc1nw_422_ntsc	109,486,080	146,642,766	**33.94%**	82.11%	76.04%	**7.39%**	36.67	53.74	**46.55%**
washdc_422_ntsc	109,486,080	163,223,193	**49.08%**	82.11%	73.33%	**10.69%**	36.48	53.62	**46.99%**
AVE			**26.46%**	**86.27%**	**83.79%**	**3.06%**			**20.17%**

TABLE 4. RESULTS OF *NON-ROI* VIDEOS

Videos without ROI	Truncated bits			Normalized power consumption			PSNR loss (dB)		
	Ref. [7]	Proposed	**diff**	Ref. [7]	Proposed	**diff**	Ref. [7]	Proposed	**diff**
bus_cif	30,412,800	43,042,560	**41.53%**	86.60%	82.47%	4.77%	48.15	40.82	**7 dB**
galleon_422_cif	72,990,720	102,643,456	**40.63%**	86.60%	83.23%	3.89%	48.20	40.72	**7 dB**
highway_cif	405,504,000	569,610,752	**40.47%**	86.60%	83.25%	3.87%	48.24	41.14	**7 dB**
tempete_cif	52,715,520	77,495,808	**47.01%**	86.60%	83.12%	4.01%	48.12	40.81	**7 dB**
bridge_far_cif	638,972,928	958,070,016	**49.94%**	82.11%	73.18%	10.88%	42.56	40.60	**2 dB**
city_4cif	182,476,800	271,175,040	**48.61%**	82.11%	73.42%	10.59%	42.48	40.84	**2 dB**
coastguard_cif	91,238,400	120,874,752	**32.48%**	82.11%	76.30%	7.08%	42.48	41.07	**1 dB**
container_cif	91,238,400	125,445,888	**37.49%**	82.11%	75.41%	8.17%	42.45	41.01	**1 dB**
flower_cif	76,032,000	107,439,360	**41.31%**	82.11%	74.72%	9.00%	42.50	40.93	**2 dB**
flower_garden_422_cif	109,486,080	162,798,336	**48.69%**	82.11%	73.40%	10.61%	42.57	40.62	**2 dB**
garden_sif	34,974,720	52,448,256	**49.96%**	82.11%	73.18%	10.88%	42.52	40.73	**2 dB**
husky_cif	76,032,000	111,882,240	**47.15%**	82.11%	73.68%	10.27%	42.48	40.91	**2 dB**
mobile_cif	91,238,400	136,164,864	**49.24%**	82.11%	73.31%	10.73%	42.57	40.70	**2 dB**
waterfall_cif	79,073,280	118,609,920	**50.00%**	82.11%	73.17%	10.89%	42.40	40.63	**2 dB**
AVE			**44.61%**	**83.40%**	**76.56%**	**8.26%**			**3dB**

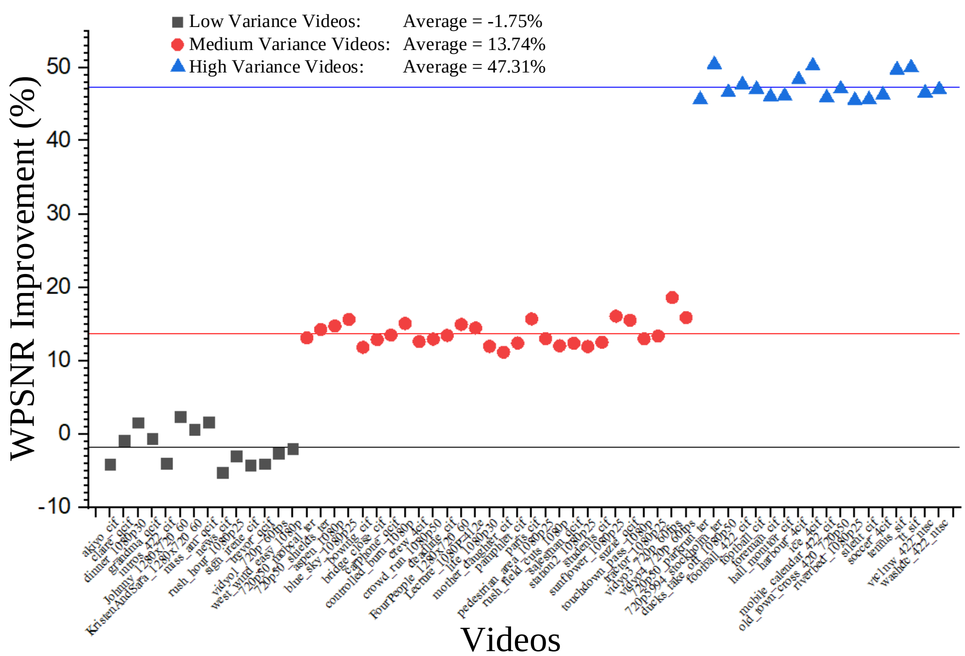


FIGURE 14. Impact of the video content characteristics on the effectiveness of the proposed technique, compared to old technique.

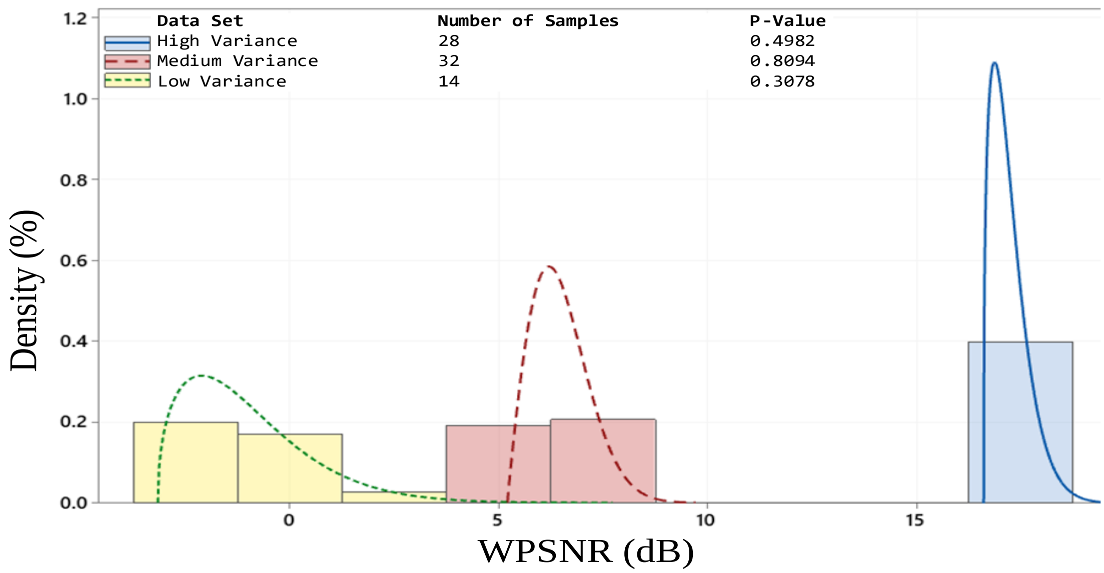


FIGURE 15. Histogram of quality Improvement distributions. Number of data points and P-value shown, between the truncation method in [7] and the proposed method. All distributions are 3-parameter Weibull distributions that fall within a 95% Confidence Interval.

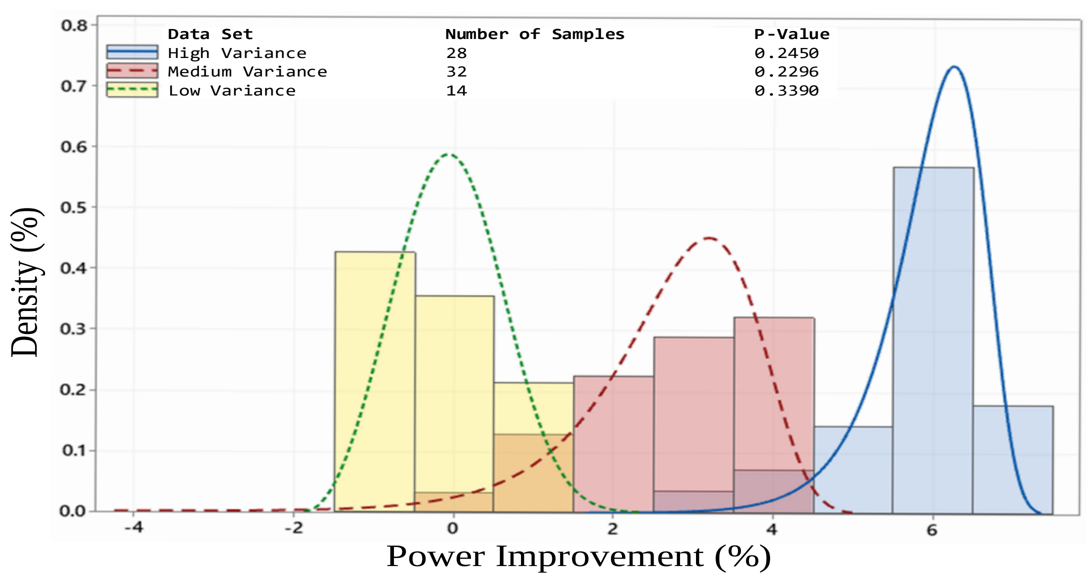


FIGURE 16 Histogram of power savings, measured in percentage improvement, between the truncation method in [7] and the proposed method. All distributions are 3-parameter Weibull distributions that fall within a 95% Confidence Interval.

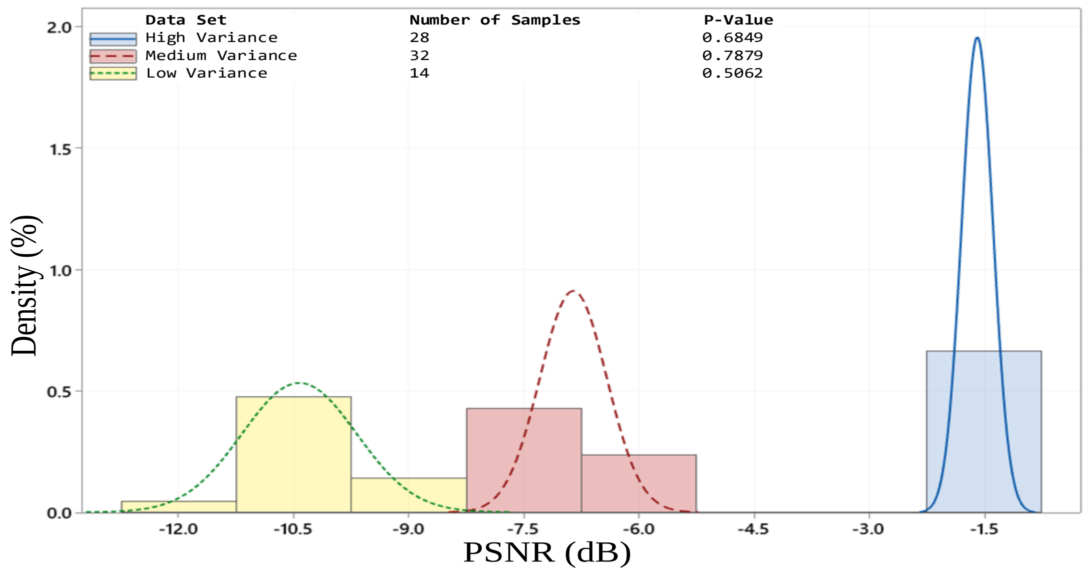


FIGURE 17. Histogram of PSNR noise increase, between the truncation method in [7] and the proposed method. All distributions are Normal Distributions that fall within a 95% Confidence Interval.

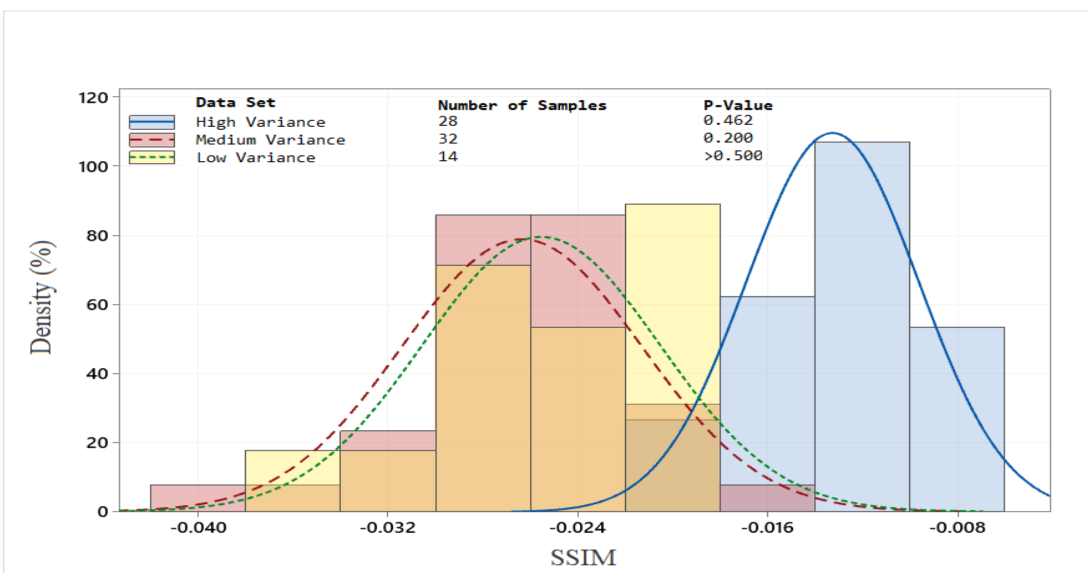


FIGURE 18. Histogram of SSIM noise increase, measured in between the truncation method in [7] and the proposed method. All distributions are 3-parameter Weibull distributions that fall within a 95% Confidence Interval.

CHAPTER VIII

COMPARISON WITH PRIOR WORK

Table 5 compares this work against state-of-the art low-power video memory designs. As shown, the proposed memory enables more-flexible run-time power-quality adaptation according to video content characteristics of each frame, while considering the important region within one frame from a perceptual point of view.

8.1 Compared to State-of-the-Art Approximate Video Memories

To enhance the power efficiency of video storage, approximate video-specific memories have been developed to store the MSBs of video data in more robust memory bitcells, such as more-than-6T SRAM bitcells [8, 9], upsized 6T [10], which usually brings implementation overhead. In order to minimize the implementation cost, those techniques typically store LSBs in error-prone but area-efficient bitcells (e.g., basic 6T [8, 10]), thereby leading to a tolerable output quality degradation with power reduction. However, for those techniques, the achieved video quality is fixed during design-time, so they lack of adaptation at run-time to meet different requirements of a variety of video applications.

8.2 Compared to State-of-the-Art Adaptive Video SRAM

To enable run-time power-quality adaptation, recently, several video SRAM designs have been presented, such as data-dependent memory [39], SRAM with selective hamming (15,11) [11], and SRAM with error-correction-code (ECC) adaptation [40]. The data-dependent SRAM consists of 10T bitcells and associated conditional pre-charge circuitry to adapt to the stored data's statistical dependencies. SRAM with selective hamming (15,11) [11] can switch between no ECC and hamming (15,11) based on the quality targets of the applications. The SRAM with ECC adaptation [40] supports three power-quality tradeoff levels, hamming code-74, hamming code-1511, and no ECC. However, those memory designs focus on hardware-level quality optimization, without considering the viewer's experience, and therefore they may cause large and inefficient design margins.

TABLE 5 COMPARISION WITH PRIOR WORK

	6T/8T SRAM [8]	Heterogeneous sizing SRAM [10]	Split-data SRAM [9]	Data-dependent SRAM [39]	SRAM with hamming [11]	SRAM with ECC [40]	Viewer-aware memories [4, 5, 6]	Content-aware memory [7]	**This Work**
Quality runtime adaptation	No	No	Yes	Yes	Yes	Yes	Yes	Yes	**Yes**
Considering viewer's experience	No	No	No	No	No	No	Yes	Yes	**Yes**
Video content adaptation	No	No	No	No	No	No	No	Yes	**Yes**
ROI awareness	No	No	No	No	No	No	No	No	**Yes**
Induced bitcell area overhead	Yes (6T and 8T)	Yes (Larger 6T)	Yes (8T and 10T)	No	Yes (10T)	No	No	No	**No**

8.3 Compared to State-of-the-Art Viewer Aware Video Memory

By introducing viewer's experience to video memory design process, a study was conducted that showed that memory failures can be leveraged to improve video system power efficiency without sacrificing viewer's experience [4, 5, 6]. The basic idea is that in high noise-tolerance viewing contexts with high-illuminance levels, memory failures are intentionally introduced by adaptively disabling LSBs of the video data stored in memories. This line of studies illustrates a new dimension of power savings for hardware design through the introduction of memory failures. However, those designs did not consider the variance of different videos and they are not sufficient to support videos with various content characteristics.

8.4 Compared to State-of-the-Art Content-Aware Video Memory

The content-aware SRAM presented in [7] is another recent viewer-aware memory design that can enable run-time power-quality adaptation based on the video content characteristics of the applications. However, it adapts the number of truncated LSBs of video data based on the average plain macroblock percentage of an entire video sample, so it is not suitable for the videos with frame-level difference. Figure 13 also compares the video output quality of the proposed memory and memory presented in [7]. As shown, the proposed memory enables more truncated bits and more power savings.

8.5 Comparison Summary

In the developed video memory technique, the ROI is identified and utilized to enable intelligent tradeoff between video quality and power efficiency of video storage in mobile devices. Accordingly, the proposed memory enables run-time quality adaptation with significantly reduced pixel bits and further power savings, as compared to existing techniques. To the best of my knowledge, this is the first work that can adapt the video storage to frame-level video content and important region from viewer's perceptual experience point of view. The proposed ROI-aware video memory is orthogonal to existing viewer-aware or data-dependent schemes and therefore can be simultaneously utilized to further optimize power efficiency.

CHAPTER IX

SUMMARY AND FUTURE WORK

9.1 Thesis Summary

In this thesis, a video content-adaptable Region-of-Interest (ROI)-aware video storage technique to optimize the power efficiency was presented. The ROI of videos is identified and protected to preserve the video quality, while other regions are truncated with 3-LSB truncation for power savings. To support the proposed method, a low-power frame buffer was developed that implemented 3-LSB truncation which enabled runtime quality and power adaptation. The results show that the proposed technique only uses 83.79% and 76.56% of the power on average for videos with ROI and without ROI respectively, as compared to the traditional memory and the state-of-the art [9], respectively. Meanwhile, the proposed technique can increase the quality (i.e. WPSNR values) by 20.17% on average for the videos with ROI and 26.46% additional truncated bits as compared to [9]. For the videos without ROI, the proposed technique can realize 44.61% additional truncated bits and 8.26% power savings as compared to [9], while maintaining a healthy above 40dB PSNR and 0.95 SSIM. This thesis focuses on the facial features as ROI of videos;

9.2 Future Work

Future investigations would include extensions of ROI, finding a general solution for ROI extraction for all video types will expand the fesability of this technology to all types of video streams. The possibility of using multiple ROI extraction algorithms to determin various types of ROIs will also be explored. Additionally, psychological experiments will be conducted to access the visual experience of viewers for hardware optimization. Another point of investigation will be to adapt the transmission protocol to include useful information within the bits that have been truncated, similar to many Steganography techniques.

REFERENCES

[1] [Online]. Available: https://www.adcolony.com/blog/2019/03/05/video-on-track-to-be-nearly-80-of-mobile-data-traffic-by-2022/.

[2] T. Liu, S. Wang, W. Lee, J. Yang, K. Hou, Lee and C, "A 125 uW, fully scalable MPEG-2 and H.264/AVC video decoder for mobile applications," *IEEE J. Solid-State Circuits,* vol. 42, no. 1, p. 161–169, 2007.

[3] F. Sampaio, M. Shafique, B. Zatt, S. Bampi and J. Henkel, "Energy-Efficient Architecture for Advanced Video Memory," in *IEEE/ACM International Conference on Computer-Aided Design (ICCAD),* 2014.

[4] D. Chen, X. Wang, J. Wang and N. Gong, "VCAS: Viewing context aware power-efficient mobile video embedded memory," in *28th IEEE Internatonal System-on-Chip Conference (SOCC),* Beijing, 2015.

[5] D. Chen, J. Edstrom, L. Yang, M. E. McCourt, J. Wang and N. Gong, "Viewer-Aware Intelligent Efficient Mobile Video Embedded Memory," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.,* vol. 26, pp. 684-696, 2018.

[6] J. Edstrom, D. Chen, J. Wang, H. Gu, E. A. Vazquez, M. E. McCourt and N. Gong, "Luminance-Adaptive Smart Video Storage System," in *International Symposium on Circuits and Systems (ISCAS),* 2016.

[7] J. Edstrom, Y. Gong, A. Haidous, B. Humphrey, M. E. McCourt, Y. Xu, J. Wang and N. Gong, "Content-Adaptive Memory for Viewer-Aware Energy-Quality Scalable Mobile Video Systems," *IEEE Access,,* vol. 7, pp. 47479-47493, 2019.

[8] I. Chang, D. Mohapatra and K. Roy, "A Priority-Based 6T/8T Hybrid SRAM Architecture for Aggressive Voltage Scaling in Video Applications," *IEEE. Trans. Circuits and Systems for Video Technology,* pp. 101-112, 2011.

[9] N. Gong, S. Jiang, A. Challapalli, S. Fernandes and R. Sridhar, "Ultra-Low Voltage Split-data-aware Embedded SRAM for Mobile Video Applications," *IEEE Trans. on Circuits and Systems II,* vol. 59, no. 12, pp. 883-887, 2012.

[10] J. Kwon, I. Lee and J. Park, "Heterogeneous SRAM Cell Sizing for Low Power H.264 Applications," *IEEE Trans. on Circuits and Systems I,,* vol. 99, no. 2, pp. 1-10, 2012.

[11] F. Frustaci, M. Khayatzadeh, D. Blaauw, D. Sylvester and M. Alioto, "SRAM for Error-Tolerant Applications With Dynamic Energy-Quality Management in 28nm CMOS," *IEEE J. Of Solid-State Circuits,* vol. 50, no. 5, pp. 1310-1323, 2015.

[12] A. Kazimirsky, A. Teman, N. Edri and A. Fish, "A 0.65-V, 500-MHz Integrated Dynamic and Static RAM for Error Tolerant Applications," *IEEE Trans. on Very Large Scale Integration (VLSI) Systems,* vol. 25, no. 9, pp. 2411-2418, 2017.

[13] M.-C. Chi, C.-H. Yeh and M.-J. Chen, "Robust Region-of-Interest Determination Based on User Attention Model Through Visual Rhythm Analysis," *IEEE Trans. on Circuits and Systems on Vodeo Technology,* vol. 19, no. 7, pp. 1025-1038, 2009.

[14] " YouTube-8M Dataset.," 2017. [Online]. Available: https://research. google.com/youtube8m/ .

[15] I. Himawan, W. Song and D. Tjondronegoro, "Automatic Region-of-Interest Detection and Prioritisation for Visually Optimised," in *IEEE Workshop on Applications of Computer Vision (WACV)*, 2013.

[16] Y. Huo, X. Wang, P. Zhang, J. Jiang and L. Hanzo, "Unequal Error Protection Aided Region of Interest Aware Wireless Panoramic Video," *IEEE Access,* vol. 7, p. 2019, 80262-80276.

[17] Y.-F. Ma, X.-S. Hua, L. Lu and H.-J. Zhang, "A generic framework of user attention model and its application in video summarization," *IEEE Trans. Multimedia,* vol. 7, no. 5, p. 907–919, 2005.

[18] I. Culjak, D. Abram, T. Pribanic, H. Dzapo and M. Cifrek, "A brief introduction to OpenCV," in *35th International Convention MIPRO*, Opatija, 2012.

[19] X.-W. Tang, X.-L. Huang, F. Hu and Q. Shi, "Human-Perception-Oriented Pseudo Analog Video Transmissions With Deep Learning," *IEEE Transactions on Vehicular Technology*, 2020.

[20] M. Shafique, "Application-guided power-efficient fault tolerance for H.264 context adaptive variable length coding," *IEEE Trans. Comput.,* vol. 66, no. 4, pp. 560-574, 2017.

[21] M. Shafique, B. Molkenthin and J. Henkel, "An HVS-based adaptive computational complexity reduction scheme for H.264/AVC video encoder using prognostic early mode exclusion," in *Design, Automation & Test in Europe Conference & Exhibition*, 2010.

[22] Y. Liang, H. Wang and K. El-Maleh, "Design and implementation of content-adaptive background skipping for wireless video," *IEEE International Symposium on Circuits and Systems,* pp. 4-7, 2006.

[23] M. Khan, S. Chakraborty, R. Astya, S. Khepra, "Face Detection and Recognition Using OpenCV," *IEEE International Conference on Computting, Communication, and Intelligent Systems (ICCCIS),* pp 116-119, 2019, .

[24] "Open_Images_v4" [Online] Available: https://www.tensorflow.org/datasets/catalog/open_images_v4.

[25] R. P. Foundation, "Raspberry Pi Documentation," [Online]. Available: https://www.raspberrypi.org/documentation/.

[26] Xilinx, "Z-turn Board (with Zynq-7020)," [Online]. Available: https://www.xilinx.com/products/boards-and-kits/1-571ww1.html. [Accessed 01 11 2020].

[27] MAGEWELL, "USB Capture Utility V3," [Online]. Available: http://www.magewell.com/usb-capture-utility-v3. [Accessed 01 11 2020].

[28] Ylonen, T. Rinne and Tatu, "scp(1) - Linux man page," 14 04 2013. [Online]. Available: https://linux.die.net/man/1/scp. [Accessed 01 11 2020].

[29] Q. Bin, "Osen Logic OSD10 h.264 decoder," [Online]. Available: http://bbs.eetop.cn/viewthread.php?tid=628991. [Accessed 2018].

[30] I. E. Richardson, The H.264 Advanced Video Compression Standard (Second Edition), West Sussex, UK: John Wiley & Sons, Ltd, 2010.

[31] Y. Wang, S. Inguva and B. Adsumilli, "YouTube UGC Dataset for Video Compression Research," in *2019 IEEE 21st International Workshop on Multimedia Signal Processing (MMSP)*, Kuala Lumpur, Malaysia, 2019.

[32] "Xiph.org Video Test Media [derf's collection]," Xiph, [Online]. Available: https://media.xiph.org/video/derf/. [Accessed 18 October 2020].

[33] Z. Wang, A. C. Bovik, H. R. Sheikh, Simoncelli and E. P., "Image quality assessment: from error visibility to structural similarity," *IEEE Trans. on Image Processing*, vol. 13, no. 4, pp. 600-612, 2004.

[34] J. Erfurt, C. R. Helmrich, S. Bosse, H. Schwarz, D. Marpe and T. Wiegand, "A Study of the Perceptually Weighted Peak Signal-To-Noise Ratio (WPSNR) for Image Compression," in *2019 IEEE International Conference on Image Processing (ICIP)*, Taipei, Taiwan, 2019 .

[35] Xilinx, "Vivado Design Suite," 2019. [Online]. Available: https://www.xilinx.com/products/design-tools/vivado.html.

[36] "FreePDK45," [Online]. Available: https://www.eda.ncsu.edu/wiki/FreePDK45:Contents.

[37] K. Pearson, "Chapter 56 - Karl Pearson, paper on the chi square goodness of fit test (1900)," in *Landmark Writings in Western Mathematics 1640-1940*, ELSEVIER, 2005, pp. 724-731.

[38] MathWave Technologies, *EasyFit Software,* 2015.

[39] C. Duan, A. J. Gotterba, M. E. Sinangil and A. P. Chandrakasan, "Energy-Efficient Reconfigurable SRAM: Reducing Read Power Through Data Statistics," *IEEE Journal Of Solid-State Circutis*, vol. 52, no. 10, pp. 2703-2711, 2017.

[40] H. Das, A. A. Haidous, S. C. Smith and N. Gong, "Flexible Low-Cost Power-Efficient Video Memory with ECC-Adaptation," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems,* 2021.

APPENDICES

Appendix A1: ROI Video Metrics (cont. 1/2)

Video Metrics For Videos With ROI	OLD CIRCUIT (OC) [7]			NEW CIRCUIT (NC)		
	SSIM	PSNR	WPSNR	SSIM	PSNR	WPSNR
akiyo_cif	0.9982	52.89	57.53	0.9703	41.96	55.14
claire_qcif	0.9981	53.36	57.61	0.9792	42.77	57.10
dinner_1080p30	0.9982	52.78	57.32	0.9766	43.28	58.21
grandma_qcif	0.9984	52.91	57.57	0.9774	43.03	57.21
intros_422_cif	0.9985	52.89	57.45	0.9686	41.71	55.15
Johnny_1280x720_60	0.9980	52.96	57.07	0.9780	43.29	58.44
KristenAndSara_1280x720_60	0.9980	52.91	56.99	0.9745	42.75	57.37
miss_am_qcif	0.9978	52.90	57.60	0.9762	43.88	58.54
news_cif	0.9985	52.90	57.52	0.9710	41.40	54.48
rush_hour_1080p25	0.9980	52.88	57.49	0.9621	42.03	55.78
sign_irene_cif	0.9983	52.89	57.48	0.9703	41.69	55.01
trevor_qcif	0.9986	52.53	57.31	0.9729	41.66	54.99
vidyo1_720p_60fps	0.9981	52.90	57.54	0.9658	42.12	56.05
west_wind_easy_1080p	0.9987	51.70	56.81	0.9760	41.90	55.68
720p50_mobcal_ter	0.9945	48.11	47.88	0.9672	41.14	54.16
720p50_shields_ter	0.9951	48.11	47.69	0.9713	41.23	54.48
aspen_1080p	0.9936	48.15	47.92	0.9692	41.38	54.97
blue_sky_1080p25	0.9929	48.11	47.66	0.9749	41.70	55.10
bowing_cif	0.9922	47.97	48.02	0.9678	41.24	53.69
bridge_close_cif	0.9962	48.36	47.95	0.9667	41.11	54.13
carphone_qcif	0.9931	48.10	48.04	0.9709	41.74	54.52
controlled_burn_1080p	0.9939	48.16	47.96	0.9690	41.50	55.18
crew_4cif	0.9938	48.13	47.54	0.9648	40.84	53.54
crowd_run_1080p50	0.9957	48.13	47.42	0.9687	40.84	53.55
deadline_cif	0.9951	48.15	48.02	0.9791	41.75	54.48
FourPeople_1280x720_60	0.9920	48.10	47.96	0.9660	41.65	55.12
Lecture_1080P-412e	0.9900	48.15	48.07	0.9665	42.64	57.02
life_1080p30	0.9938	48.25	48.04	0.9680	41.51	54.99
mother_daughter_cif	0.9925	47.99	48.03	0.9642	41.14	53.78
pamphlet_cif	0.9940	47.92	47.93	0.9657	40.84	53.27
paris_cif	0.9960	48.15	47.88	0.9773	41.04	53.81
pedestrian_area_1080p25	0.9920	48.43	48.05	0.9632	41.88	55.59
rush_field_cuts_1080p	0.9932	48.14	47.86	0.9634	41.12	54.09
salesman_qcif	0.9946	48.13	47.97	0.9657	41.03	53.74
station2_1080p25	0.9923	48.14	47.79	0.9550	40.92	53.71
students_cif	0.9941	48.14	47.92	0.9701	40.98	53.64
sunflower_1080p25	0.9921	48.13	47.81	0.9602	41.03	53.79

FIGURE A1. ROI Video Metrics for all videos presented in paper. This is the raw results of each video using the proposed and old circuitry [7]. Videos with a WPSNR of 'nan' are videos that do not contain a single ROI.

Appendix A1: ROI Video Metrics (CONT. 2/2)

Video Metrics For Videos With ROI	OLD CIRCUIT (OC) [7]			NEW CIRCUIT (NC)		
	SSIM	PSNR	WPSNR	SSIM	PSNR	WPSNR
sunflower_1080p25	0.9921	48.13	47.81	0.9602	41.03	53.79
suzie_qcif	0.9918	48.12	48.07	0.9709	42.57	55.79
touchdown_pass_1080p	0.9910	48.14	47.94	0.9617	41.95	55.38
tractor_1080p25	0.9936	48.11	47.50	0.9626	40.96	53.67
vidyo3_720p_60fps	0.9911	48.12	48.11	0.9630	41.35	54.54
vidyo4_720p_60fps	0.9916	48.19	48.13	0.9686	42.11	55.78
720p50_parkrun_ter	0.9909	42.47	36.71	0.9727	40.80	53.47
720p5994_stockholm_ter	0.9800	42.47	35.49	0.9638	40.77	53.38
ducks_take_off_1080p50	0.9894	42.49	36.36	0.9760	40.72	53.30
football_422_cif	0.9832	42.49	36.54	0.9709	40.96	53.96
football_cif	0.9818	42.44	36.58	0.9703	40.95	53.79
foreman_cif	0.9800	42.48	37.01	0.9695	41.32	54.04
hall_monitor_cif	0.9743	42.50	36.66	0.9568	40.84	53.56
harbour_4cif	0.9946	42.50	36.34	0.9790	41.00	53.91
ice_4cif	0.9703	41.94	35.22	0.9619	40.66	52.91
mobile_calendar_422_cif	0.9915	42.46	36.37	0.9806	40.59	53.06
old_town_cross_420_720p50	0.9751	42.48	36.44	0.9583	40.87	53.60
riverbed_1080p25	0.9765	42.48	36.62	0.9575	40.71	53.29
silent_cif	0.9820	42.48	36.70	0.9689	40.94	53.46
soccer_4cif	0.9783	42.49	36.49	0.9667	40.93	53.36
tennis_sif	0.9796	42.50	36.27	0.9676	41.08	54.29
tt_sif	0.9805	42.48	36.16	0.9677	41.09	54.24
vtc1nw_422_ntsc	0.9774	42.37	36.67	0.9678	41.09	53.74
washdc_422_ntsc	0.9883	42.40	36.48	0.9774	40.91	53.62
tempete_cif	0.9966	48.12	nan	0.9779	40.81	nan
galleon_422_cif	0.9956	48.20	nan	0.9707	40.72	nan
highway_cif	0.9925	48.24	nan	0.9536	41.14	nan
bus_cif	0.9970	48.15	nan	0.9725	40.82	nan
bridge_far_cif	0.9660	42.56	nan	0.9453	40.60	nan
city_4cif	0.9869	42.48	nan	0.9702	40.84	nan
coastguard_cif	0.9877	42.48	nan	0.9735	41.07	nan
container_cif	0.9766	42.45	nan	0.9669	41.01	nan
flower_cif	0.9929	42.50	nan	0.9844	40.93	nan
flower_garden_422_cif	0.9894	42.57	nan	0.9782	40.62	nan
garden_sif	0.9911	42.52	nan	0.9813	40.73	nan
husky_cif	0.9949	42.48	nan	0.9784	40.91	nan
mobile_cif	0.9925	42.57	nan	0.9855	40.70	nan
waterfall_cif	0.9883	42.40	nan	0.9775	40.63	nan

Appendix A2: Main.py (cont. 1/2)

```
#written by Liam Oswald
import struct
import sys
import math
import pickle
import cv2
import numpy as np
from tqdm import tqdm
import os
from skimage.measure import compare_ssim
import Macroblock
import OpenCVROI
import MBTruncation
import FlatTruncation
import ROIMBTruncation
import pulldatatocsv
import WeightedPSNR

def FlatTruncateAmountViaFile(FileName):
    fin = open(str(FileName), 'r')
    list = str(fin.read()).split(',')
    MBpercent = list[19]
    fin.close()

    if float(MBpercent) >= 21.5571:
        i = 1
    elif float(MBpercent) >= 1.96405:
        i = 2
    else:
        i = 3
    return i

VideoRepositoryDir = '/home/student/Desktop/Duplicate/vid'
#VideoRepositoryDir = '/home/student/Desktop/Duplicate/D3/Videos'
#VideoRepositoryDir = '/home/student/PycharmProjects/H264Project/Videos'
ResultsRepositoryDir = '/home/student/PycharmProjects/H264Project/Results'
#here

# PixelWidth = 1920
# PixelHight = 1080
# FPS = 25
# FrameCount = 600

#
PixelWidth = 352
PixelHight = 288
FPS = 24
FrameCount = 300
```

FIGURE A2. Main.py, This Main Function calls all other scripts. Written by William Oswald, using Python 2.7. The input is a YUV422 video and produces as outputs the digital results of the proposed and previous circuits, calculates PSNR, SSIM and WPSNR for both. It also saves each from of the output video as a .png file.

Appendix A2: Main.py (cont. 2/2)

```
def main():
    VideoRepository = os.listdir(VideoRepositoryDir)
    print "Videos To Process: ", VideoRepository

    for Video in VideoRepository:
        VideoIn = str(VideoRepositoryDir + '/' + Video)
        ResultsOut = str(ResultsRepositoryDir + '/' + Video[:-4])
        cap = Macroblock.VideoCaptureYUV(VideoIn, (PixelHight, PixelWidth))
        FrameCount = cap.framecount

        temp_folder = os.path.join(os.getcwd(), 'temp')

        frames_path = os.path.join(os.getcwd(), 'Results', str(VideoIn[:-4].rsplit('/', 1)[-
1]) + "/frames/")
        MBframe_path = os.path.join(os.getcwd(), 'Results', str(VideoIn[:-4].rsplit('/', 1)[-
1]) + "/MBTruncation_Results/")

        cap = Macroblock.VideoCaptureYUV(VideoIn, (PixelHight, PixelWidth))
        ROIMBTruncation.OpenCVMBTruncate(cap, FPS, ResultsOut, 1.25)
        ROIMBT_dir = str(ResultsOut + '/ROIMBTruncation/' + str(1.25) +
'/ROIMBTruncation_output.yuv')
        Macroblock.Main(VideoIn, ROIMBT_dir, cap.width, cap.height, str(ResultsOut +
'/ROIMBTruncation/' + str(1.25) + '/ROIMBTruncation_output/'))

        n = FlatTruncateAmountViaFile(str(ResultsOut + '/ROIMBTruncation/' + str(1.25) +
'/ROIMBTruncation_output/' + Video[:-4] + '.csv'))

        cap = Macroblock.VideoCaptureYUV(VideoIn, (PixelHight, PixelWidth))
        FlatTruncation.FlatTruncate(cap, str(ResultsOut + '/flat/'), FPS, n)
        Macroblock.Main(VideoIn, str(ResultsOut + '/flat/' + str(n) + '_OUTPUT.yuv'),
cap.width, cap.height, str(ResultsOut + '/flat/' + str(n) + '/'))

        Old_method_Frame = str(ResultsOut + '/flat/' + str(n) + '_OUTPUT.yuv')
        WeightedPSNR.Main(VideoIn, ROIMBT_dir, Old_method_Frame, cap.width, cap.height,
str(ResultsOut + '/WeightedPSNR/' + str(1.25) + '/'), alpha=0.90)


        pullFile = '/home/student/PycharmProjects/H264Project/Results/' + Video[:-4] + '/'
        pulldatatocsv.main(pullFile, PixelWidth, PixelHight, n, FrameCount)


        cap.release()



if __name__ == "__main__":
    main()
```

Appendix A3: Macroblock.Py (cont. 1/5)

```
#!/usr/bin/python
#Script written by Ali Haidous, modified by Liam Oswald

import struct
import sys
import math
import pickle
import cv2
import numpy as np
from tqdm import tqdm
import os
from skimage.measure import compare_ssim

FIRST_PLAIN_MB = 21.5571
SECOND_PLAIN_MB = 1.96405
class VideoCaptureYUV(object):
    def __init__(self, filename, size):
        self.height, self.width = size
        self.filename = filename
        self.filesize = os.stat(filename).st_size
        self.framecount = ( 2 * self.filesize ) / ( self.height * self.width * 3 )
        self.frame_len = self.width * self.height * 3 / 2
        self.f = open(filename, 'rb')
        self.shape = (int(self.height*1.5), self.width)

    def file_statistics(self):
        return (self.filesize, self.framecount)

    def read_raw(self):
        try:
            raw = self.f.read(self.frame_len)
            yuv = np.frombuffer(raw, dtype=np.uint8)
            yuv = yuv.reshape(self.shape)
        except Exception as e:
            print str(e)
            return False, None
        return True, yuv

    def read(self):
        ret, yuv = self.read_raw()
        if not ret:
            return ret, yuv
        bgr = cv2.cvtColor(yuv, cv2.COLOR_YUV2BGR_I420)
        return ret, bgr

    def fetch_raw_frame(self, frame_num):
        f = open(self.filename, 'rb')
        raw = None
        for _ in range(frame_num):
            raw = f.read(self.frame_len)
```

FIGURE A3. Macroblock.py, originally written by Ali Haidous. This file has been heavily modified by Williiam Oswald. This Script calculates the Macroblock variance on a per-frame basis.

Appendix A3: Macroblock.Py (cont. 2/5)

```
        try:
            yuv = np.frombuffer(raw, dtype=np.uint8)
            yuv = yuv.reshape(self.shape)
        except Exception as e:
            print str(e)
            return None
        return yuv

    def display_raw_frame(self, raw_frame, name="frame"):
        frame = cv2.cvtColor(raw_frame, cv2.COLOR_YUV2BGR_I420)
        #cv2.imshow(name, frame)
        cv2.imwrite(name, frame)

    def play_video(self):
        while True:
            ret, frame = self.read(cv2.COLOR_YUV2BGR_I420)
            if ret:
                cv2.imshow("frame", frame)
                cv2.waitKey(30)
            else:
                break

    def release(self):
        self.f.close()

class VideoWriter(object):
    def __init__(self, filename):
        self.filename = filename
        self.f = open(filename, 'wb')

    def writeFrame(self, yuvFrame):
        self.f.write(yuvFrame)

    def release(self):
        self.f.close()



def calc_macroblock_per(yuv_frame, low_variance_threshold=1.25):
    offset = int(len(yuv_frame)/3)
    rows = len(yuv_frame) - offset
    columns = len(yuv_frame[0])

    y = yuv_frame[0:rows, 0:columns]
    u = yuv_frame[rows:rows+(offset/2), 0:columns]
    v = yuv_frame[rows+(offset/2):rows+offset, 0:columns]

    def macbroblock_per(sub_frame):
        total_macroblocks = 0
        plain_macroblocks = 0
        macroblock_per = 0
        for row_position in range(0, len(sub_frame), 16):
            for column_position in range(0, len(sub_frame[0]), 16):
                macroblock = []
                total_macroblocks += 1

                for j in range(row_position, row_position+16):
                    for i in range(column_position, column_position+16):
                        try:
                            macroblock.append(0.0001560911143834408 *
pow(int(sub_frame[j][i]), 2.628389343175764))
                        except IndexError:
                            break
```

Appendix A3: Macroblock.Py (cont. 3/5)

```
        if ret:
            (y_per, u_per, v_per) = calc_macroblock_per(frame)
            #print (y_per, u_per, v_per)
            macroblock_per_y.append(y_per)
            macroblock_per_u.append(u_per)
            macroblock_per_v.append(v_per)
        else:
            break

    return (macroblock_per_y, macroblock_per_u, macroblock_per_v)


#This main function has been heavily modified by Liam Oswald.
def Main(VideoOld, VideoNew, xRez, yRez, path):
    # Get arguments
    filename = VideoOld
    xres = int(xRez)
    yres = int(yRez)

    frames_path = path + 'frames/'

    # Do OS operations
    #path = os.path.join(os.getcwd(), str(filename[:-4].rsplit('/', 1)[-1]))
    #path = os.path.join(os.getcwd(),'Results', str(filename[:-4].rsplit('/', 1)[-1]))
    #frames_path = os.path.join(os.getcwd(), str(filename[:-4].rsplit('/', 1)[-1])+"/frames/")
    try:
        #os.makedirs(path)
        os.makedirs(frames_path[:-1])
    except OSError:
        print ("Creation of the directory %s failed" % frames_path)
    else:
        print ("Successfully created the directory %s " % frames_path)

    # Read in the file
    cap = VideoCaptureYUV(filename, (yres, xres))
    filesize, framecount = cap.file_statistics()

    capNew = VideoCaptureYUV(VideoNew, (yres, xres))
    filesizeNew, framecountNew = capNew.file_statistics()

    print "Size of file in bytes: %d\nNumber of frames: %d\n" % (filesize, framecount)
    #cap.play_video()

    # Calculate macroblock percentage per frame
    (macroblock_per_y, macroblock_per_u, macroblock_per_v) =
CalculateMacroblockPercentage(cap, framecount)
    (macroblock_per_y_new, macroblock_per_u_new, macroblock_per_v_new) =
CalculateMacroblockPercentage(capNew, framecountNew)

    # Calculate average macroblock percentages
    macroblock_per_y_avg = sum(macroblock_per_y) / len(macroblock_per_y)
    macroblock_per_u_avg = sum(macroblock_per_u) / len(macroblock_per_u)
    macroblock_per_v_avg = sum(macroblock_per_v) / len(macroblock_per_v)
    macroblock_per_y_avg_new = sum(macroblock_per_y_new) / len(macroblock_per_y_new)
    macroblock_per_u_avg_new = sum(macroblock_per_u_new) / len(macroblock_per_u_new)
    macroblock_per_v_avg_new = sum(macroblock_per_v_new) / len(macroblock_per_v_new)
    #Filter >= 99% macroblocks
    for x in range(len(macroblock_per_y)):
        if macroblock_per_y[x] >= 99:
            macroblock_per_y[x] = macroblock_per_y_avg
    for x in range(len(macroblock_per_u)):
        if macroblock_per_u[x] >= 99:
            macroblock_per_u[x] = macroblock_per_u_avg
```

Appendix A3: Macroblock.Py (cont. 4/5)

```
    for x in range(len(macroblock_per_v)):
        if macroblock_per_v[x] >= 99:
            macroblock_per_v[x] = macroblock_per_v_avg
    #Filter >= 99% for new video as well
    for x in range(len(macroblock_per_y_new)):
        if macroblock_per_y_new[x] >= 99:
            macroblock_per_y_new[x] = macroblock_per_y_avg_new
    for x in range(len(macroblock_per_u_new)):
        if macroblock_per_u_new[x] >= 99:
            macroblock_per_u_new[x] = macroblock_per_u_avg_new
    for x in range(len(macroblock_per_v_new)):
        if macroblock_per_v_new[x] >= 99:
            macroblock_per_v_new[x] = macroblock_per_v_avg_new
    # Calculate average psnr and ssim for the whole video
    cap = VideoCaptureYUV(filename, (yres, xres))
    capNew = VideoCaptureYUV(VideoNew, (yres, xres))
    psnr_old = []
    ssim_old = []
    psnr_new = []
    ssim_new = []
    for index in tqdm(range(framecount), unit="MacroBlock PSNR and SSIM Calc"):
        try:
            ret, frame = cap.read_raw()
            retNew, frameNew = capNew.read_raw()
        except Exception:
            break
        if (ret & retNew):
            frame_truncate_old = frame
            frame_truncate_new = frameNew

            psnr_old.append(psnr(frameNew, frame_truncate_old))
            (ssim, _) = compare_ssim(frameNew, frame_truncate_old, full=True)
            ssim_old.append(ssim)
            # psnr_new.append(psnr(frame, frame_truncate_new))
            # (ssim, _) = compare_ssim(frame, frame_truncate_new, full=True)
            # ssim_new.append(ssim)
            psnr_new.append("Liam Lazy")
            ssim_new.append("Liam Lazy")
            #cap.display_raw_frame(frame, frames_path+str(filename[:-4].rsplit('/', 1)[-
1])+"_frame"+str(index)+".png")
            cap.display_raw_frame(frame_truncate_old, frames_path+str(filename[:-
4].rsplit('/', 1)[-1])+"_Original"+str(index)+".png")
            capNew.display_raw_frame(frame_truncate_new, frames_path+str(filename[:-
4].rsplit('/', 1)[-1])+"_New"+str(index)+".png")
        else:
            break
    # Calculate max and min macroblock frame index for y u and v
    max_frame_y_index = macroblock_per_y.index(max(macroblock_per_y))
    min_frame_y_index = macroblock_per_y.index(min(macroblock_per_y))
    max_frame_u_index = macroblock_per_u.index(max(macroblock_per_u))
    min_frame_u_index = macroblock_per_u.index(min(macroblock_per_u))
    max_frame_v_index = macroblock_per_v.index(max(macroblock_per_v))
    min_frame_v_index = macroblock_per_v.index(min(macroblock_per_v))
    psnr_old_avg = sum(psnr_old) / len(psnr_old)
    ssim_old_avg = sum(ssim_old) / len(ssim_old)

    # Write data to CSV file
    with open(os.path.join(path,str(filename[:-4].rsplit('/', 1)[-1])+".csv"), "wb") as file:
        file.write("Original Video,"+
                   "New Video,"+
                   ","+
                   "MB % Y Avg:,"+
                   "MB % U Avg:,"+
                   "MB % V Avg:,"+
```

Appendix A3: Macroblock.Py (cont. 5/5)

```
                   "MB Y Max Idx:,"+
                   "MB Y Min Idx:,"+
                   "MB U Max Idx:,"+
                   "MB U Min Idx:,"+
                   "MB V Max Idx:,"+
                   "MB V Min Idx:,"+
                   "PSNR Avg,"+
                   "SSIM Avg:,"+
                   #"PSNR New Avg:,"+
                   #"SSIM New Avg:,"+"\n")
                   "," +
                   "," + "\n")
        file.write(str(VideoOld[:-4].rsplit('/', 1)[-1])+","+
                   str(VideoNew[:-4].rsplit('/', 1)[-1])+","+
                   str(',')+
                   str(macroblock_per_y_avg)+","+
                   str(macroblock_per_u_avg)+","+
                   str(macroblock_per_v_avg)+","+
                   str(max_frame_y_index)+","+
                   str(min_frame_y_index)+","+
                   str(max_frame_u_index)+","+
                   str(min_frame_u_index)+","+
                   str(max_frame_v_index)+","+
                   str(min_frame_v_index)+","+
                   str(psnr_old_avg)+","+
                   str(ssim_old_avg)+","+
                   #str(psnr_new_avg)+","+
                   #str(ssim_new_avg)+"\n\n\n")
                   str('') + "\n\n\n")
        file.write("Frame,MB % Y,MB % U,MB % V,PSNR ,SSIM, \n")
        for index, mb_y, mb_u, mb_v, psnr_o, ssim_o, in zip(range(framecount),
                                                            macroblock_per_y,
                                                            macroblock_per_u,
                                                            macroblock_per_v,
                                                            psnr_old,
                                                            ssim_old,
                                                            #psnr_new,
                                                            #ssim_new):
                                                            ):

#file.write(str(index)+","+str(mb_y)+","+str(mb_u)+","+str(mb_v)+","+str(psnr_o)+","+str(ssim_
o)+","+str(psnr_n)+","+str(ssim_n)+"\n")
            file.write(str(index) + "," + str(mb_y) + "," + str(mb_u) + "," + str(mb_v) + ","
+ str(psnr_o) + "," + str(ssim_o) + "," + "\n")


if __name__ == "__main__":

    Main()
```

Appendix A4: FlatTruncate.Py (cont. 1/2)

```
import cv2
import os
import tqdm
import numpy as np
import Macroblock

def TruncateIntValue(a, n):
    if (n == 1):
        return (int(a) & 0b11111111 | 0b00000001)
    elif (n==2):
        return (int(a) & 0b11111110 | 0b00000010)
    elif (n==3):
        return (int(a) & 0b11111100 | 0b00000100)
    elif (n==4):
        return (int(a) & 0b11111000 | 0b00001000)
    elif (n==5):
        return (int(a) & 0b11110000 | 0b00010000)
    elif (n==6):
        return (int(a) & 0b11100000 | 0b00100000)
    elif (n==7):
        return (int(a) & 0b11000000 | 0b01000000)
    elif (n==8):
        return (int(a) & 0b10000000 | 0b10000000)
    elif (n==0):
        return a

def FlatTruncate(CapturedVideo, fileOutDirectory, FPS, n=2):
    try:
        os.makedirs(fileOutDirectory[:-1])
    except OSError:
        print ("Creation of the directory failed")
    else:
        print ("Successfully created the directory")

    try:
        os.makedirs(str(fileOutDirectory) + str(n))
    except OSError:
        print ("Creation of the directory failed")
    else:
        print ("Successfully created the directory")
    cap = CapturedVideo

    print str(fileOutDirectory) + str(n)
    # Setup output mp4 video that will show blue box around ROI's

    name = str(fileOutDirectory + str(n) + '_OUTPUT.yuv')


    #Reference FOURCC values from http://www.fourcc.org/codecs.php
    Frame = Macroblock.VideoWriter(str(fileOutDirectory + str(n) + '_OUTPUT.yuv'))
    print (str(fileOutDirectory + str(n) + '_OUTPUT.yuv'))
```

FIGURE A4. FlatTruncate.py This script was written by William Oswald, this takes a YUV 422 video, and writes a flat truncated video to a desired location. For our purposes 3-bit truncation was use

Appendix A4: FlatTruncate.Py (cont. 2/2)

```
    try:


        countTotal = 0
        frameNum = 0

        for _ in tqdm.tqdm(range(int(cap.framecount)), unit="Flat Truncation"):
            frameNum += 1
            count = 0

            ret, yuvimgRaw = cap.read_raw()
            if not ret:
                break
            yuvimg = np.copy(yuvimgRaw)

            offset = int(len(yuvimgRaw) / 3)
            rows = len(yuvimgRaw) - offset
            columns = len(yuvimgRaw[0])


            # print yuvimg.shape
            #
            # cv2.imshow("image stack", yuvimg)
            # cv2.waitKey(1)
            for i in range(rows):
                for j in range(int(yuvimg.shape[1])):
                    yuvimg[i, j] = TruncateIntValue(yuvimg[i,j], n)
                    count += 1
            #r.write('Frame Number: ' + str(frameNum) + ' Truncations Preformed: ' +
str(count) + '\n')
            countTotal = countTotal + count

            # cv2.imshow("image stack", yuvimg)
            # cv2.waitKey(1)


            #print np.array_equal(yuvimg, yuvimgRaw)

            Frame.writeFrame(yuvimg)

        #r.write('Total Number of Truncations: ' + str(countTotal))
        #r.close()


    finally:
        Frame.release()
```

d.

Appendix A5: MBTruncation.py (cont. 1/2)

```
#Written by Liam Oswald
import Macroblock
import OpenCVROI
import struct
import sys
import math
import pickle
import cv2
import numpy as np
import tqdm
import os
from skimage.measure import compare_ssim

def Display2Images(img1, img2):
    stack = np.hstack((img1, img2))

    cv2.imshow("image stack", stack)
    cv2.waitKey(1)

def macroblock_per(yuv_frame, low_variance_threshold=1.25):
    offset = int(len(yuv_frame)/3)
    rows = len(yuv_frame) - offset
    columns = len(yuv_frame[0])

    y = yuv_frame[0:rows, 0:columns]
    u = yuv_frame[rows:rows+(offset/2), 0:columns]
    v = yuv_frame[rows+(offset/2):rows+offset, 0:columns]

    def macbroblock_per(sub_frame, macroblocksize, Ybox):
        visualFrame = np.copy(sub_frame)
        truncatedFrame = np.copy(sub_frame)

        bitsTruncated = 0

        for row_position in range(0, len(sub_frame), 16):
            for column_position in range(0, len(sub_frame[0]), 16):
                macroblock = []

                for j in range(row_position, row_position+16):
                    for i in range(column_position, column_position+16):
                        try:
                            macroblock.append(0.0001560911143834408 *
pow(int(sub_frame[j][i]), 2.628389343175764))
                        except IndexError:
                            break

                try:
                    avg_lum = sum(macroblock) / len(macroblock)
                    variance = sum([pow(byte - avg_lum, 2) / len(macroblock) for byte in
macroblock])
                except ZeroDivisionError:
                    pass
```

FIGURE A4. MBTruncation.py – This Script was written by William Oswald. This takes a YUV 422 video, and truncates macroblocks based on variance levels, which it calculates itself

Appendix A5: MBTruncation.py (cont. 2/2)

```
                else:
                    if variance >= low_variance_threshold:
                        if Ybox:
                            visualFrame[row_position:row_position+4,
column_position:column_position + 16] = 255
                        else:
                            visualFrame[row_position:row_position+16,
column_position:column_position+16] = 255

                        for i in range(macroblocksize):
                            for j in range(16):
                                if ((row_position + i) < len(sub_frame)):
                                    truncatedFrame[row_position + i, column_position + j] =
OpenCVROI.TruncateIntValue(truncatedFrame[row_position + i, column_position + j])
                                    bitsTruncated += 1
        return visualFrame, truncatedFrame, bitsTruncated
    yVisualFrame, yTruncatedFrame, bits1 = macbroblock_per(y, 16, True)
    uVisualFrame, uTruncatedFrame, bits2 = macbroblock_per(u, 16, False)
    vVisualFrame, vTruncatedFrame, bits3 = macbroblock_per(v, 16, False)
    visualframe = np.vstack((yVisualFrame,uVisualFrame,vVisualFrame))
    truncatedframe = np.vstack((yTruncatedFrame, uTruncatedFrame, vTruncatedFrame))
    totalbits = bits1 + bits2 + bits3
    return (visualframe, truncatedframe, totalbits)
def VisualizePlainMacroblocks(cap, FPS, saveDir, low_variance_threshold):
    try:
        newsaveDir = os.path.join(saveDir, "MBTruncation_Results/")
        os.makedirs(newsaveDir)
    except OSError:
        print ("Creation of the directory failed")
    else:
        print ("Successfully created the directory")
    #paramiters for mp4 visual output
    namemp4 = str(newsaveDir + "/MBVisualization.mp4")
    fourccmp4 = cv2.VideoWriter_fourcc(*'mp4v')
    Framemp4 = cv2.VideoWriter(namemp4, fourccmp4, FPS, (cap.width, cap.height))
    #paramiters for yuv output video with truncation
    name = str(newsaveDir + "/MBTruncation_output.yuv")
    #fourcc = cv2.VideoWriter_fourcc(*'IYUV')
    #Frame = cv2.VideoWriter(name, fourcc, FPS, (cap.width, cap.height))
    Frame = Macroblock.VideoWriter(name)

    f = open(os.path.join(saveDir, "MBTruncation_Results/", 'Macroblock_Report.txt'), 'w')
    frameNum = 0
    totalTruncations = 0

    for _ in tqdm.tqdm(range(int(cap.framecount)), unit="MBTruncation Visualize"):
        ret, yuvimgRaw = cap.read_raw()
        if not ret:
            break
        #copy yuvimg for editing
        yuvimg = np.copy(yuvimgRaw)
        visual, truncatedFrame, bitstruncated = macroblock_per(yuvimg, low_variance_threshold)
        f.write('Frame Num: ' + str(frameNum) + '  Truncations preformed: ' +
str(bitstruncated) + '\n')
        totalTruncations = totalTruncations + bitstruncated
        #save Visual into mp4 file
        Framemp4.write(cv2.cvtColor(visual, cv2.COLOR_YUV2BGR_I420))
        Frame.writeFrame(truncatedFrame)
        frameNum += 1
    Frame.release()
    Framemp4.release()

    f.write('Total Truncations Preformed: ' + str(totalTruncations))
    f.close()
```

Appendix A6: OpenCVROI.Py (cont. 1/2)

```
# Script written by William Oswald Python Version 3.6

import cv2
import os
import tqdm
import numpy as np
def TruncateIntValue(a):
    return (int(a) & 0b11111100 | 0b00000100)
def InROIRange(Pixelx, Pixely, faces):
    for (x, y, w, h) in faces:
        if ((x < Pixelx < (x + w)) and (y < Pixely < (y + h))):
            return True
        else:
            return False

def ROI_CSV_OUT(faces, fileDirecotry, f):
    # Setup CSV file to save location of the 4 corners for a ROI
    count = 0
    tempstring = ''

    for (x, y, w, h) in faces:
        tempstring = (tempstring + str(x) + ',' + str(y) + ' ' + str(x) + ',' + str(y + h) + '
' + str(
            x + w) + ',' + str(y) + ' ' + str(x + w) + ',' + str(y + h))
        count = count + 1

    f.write((str(count) + ' ' + tempstring + '\n'))
def FindFaces(RawYUVFrame, face_cascade):
    # Load the cascade
    # Convert to grayscale
    gray = cv2.cvtColor(RawYUVFrame, cv2.COLOR_BGR2GRAY)
    # Detect the faces
    faces = face_cascade.detectMultiScale(gray, 1.1, 4)
    return faces

def TranslatePositon(x,y, hight, width):
    #YUV file format explained here: https://answers.opencv.org/question/100149/how-to-get-y-
u-v-from-image/
    ypos = [x,y]
    ve = [int(hight + (x / 4)), int(y / 2)]
    vo = [int(hight + (x / 4)), int((y / 2) + (width / 2))]
    ue = [int(1.25 * hight + (x / 4)), int(y / 2)]
    uo = [int(1.25 * hight + (x / 4)), int((y / 2) + (width / 2))]
    return ypos,ve,vo,ue,uo

def OpenCVTruncate(pixelWidth=1920, pixelHight=1080, FPS=24.0, fileIn='Video.yuv',
fileDirecotry = os.getcwd(), CapturedVideo = None, frameCount = None):

    try:
        os.makedirs(os.path.join(os.getcwd(),'Results', str(fileIn[:-4].rsplit('/', 1)[-1])))
```

FIGURE A6. OpenCVROI.py – This Script was written by William Oswald. This Script uses the Hardcascade_frontalFace_mlt2.xml Neural Network model to find faces in a YUV video frame, and writes ROI locations in a table as output

Appendix A6: OpenCVROI.Py (cont. 2/2)

```
    except OSError:
        print ("Creation of the directory failed")
    else:
        print ("Successfully created the directory")
    f = open(os.path.join(os.getcwd(),'Results', str(fileIn[:-4].rsplit('/', 1)[-
1]),'ROI_Locations.csv'), 'w')
    r = open(os.path.join(os.getcwd(), 'Results', str(fileIn[:-4].rsplit('/', 1)[-1]),
'ROI_Report.txt'), 'w')
    name = str(os.path.join(os.getcwd(),'Results', str(fileIn[:-4].rsplit('/', 1)[-1]),
((fileIn[:-4].rsplit('/', 1)[-1])+'_OUTPUT.yuv')))

    face_cascade = cv2.CascadeClassifier('haarcascade_frontalface_alt2.xml')


    # To capture video from webcam.
    cap = CapturedVideo

    # Setup output mp4 video that will show blue box around ROI's
    #Reference FOURCC values from http://www.fourcc.org/codecs.php
    #fourcc = cv2.VideoWriter_fourcc(*'YV12')
    fourcc = cv2.VideoWriter_fourcc(*'IYUV')
    Frame = cv2.VideoWriter(name, fourcc, FPS, (cap.width, cap.height))

    try:
        for _ in tqdm.tqdm(range(int(frameCount)), unit="OpenCV Truncation"):
            # Read the frame

            ret, yuvimgRaw = cap.read_raw()
            if not ret:
                break
            img = cv2.cvtColor(yuvimgRaw, cv2.COLOR_YUV2BGR_I420)
            yuvimg = np.copy(yuvimgRaw)

            faces = FindFaces(img, face_cascade)


            #if an image is not in an ROI, truncate it across the YUV frame
            totalTruncatePixels = 0
            for i in range(cap.height):
                for j in range(cap.width):
                    if (not InROIRange(j, i, faces)):
                        for [h,w] in TranslatePositon(i,j,cap.height,cap.width):
                            yuvimg[h,w] = TruncateIntValue(yuvimg[h,w])
                            #yuvimg[h, w] = 255
                            totalTruncatePixels += 1
            ROI_CSV_OUT(faces, fileDirecotry, f)

            Frame.write(cv2.cvtColor(yuvimg, cv2.COLOR_YUV2BGR_I420))

        r.write(str('OpenCVROI bits Truncated    ' + str(totalTruncatePixels) + '\n'))

    finally:
        Frame.release()
        # Release the VideoCapture object

        # Release CSV save file
        r.close()
        f.close()



if __name__ == "__main__":
    OpenCVTruncate()
```

Appendix A7: ROIMBTruncation.Py (cont. 1/3)

```
import cv2
import numpy as np
import tqdm
import os
import math
import Macroblock

def TruncateIntValue(a):
    return (int(a) & 0b11111100 | 0b00000100)
    #return 0b00000000

def InROIRange(Pixelx, Pixely, faces):
    for (x, y, w, h) in faces:
        if ((x < Pixelx < (x + w)) and (y < Pixely < (y + h))):
            return True
        else:
            return False

def ROI_CSV_OUT(faces, fileDirecotry, f):
    # Setup CSV file to save location of the 4 corners for a ROI
    count = 0
    tempstring = ''

    for (x, y, w, h) in faces:
        tempstring = (tempstring + str(x) + ',' + str(y) + ' ' + str(x) + ',' + str(y + h) + '
' + str(
            x + w) + ',' + str(y) + ' ' + str(x + w) + ',' + str(y + h))
        count = count + 1
    f.write((str(count) + ' ' + tempstring + '\n'))

def FindFaces(RawYUVFrame, face_cascade):
    # Load the cascade
    # Convert to grayscale
    gray = cv2.cvtColor(RawYUVFrame, cv2.COLOR_BGR2GRAY)
    # Detect the faces
    faces = face_cascade.detectMultiScale(gray, 1.1, 4)
    return faces

def TranslatePositon(x,y, hight, width):
    ypos = [x, y]
    ve = [int((x / 2)), int(hight + (y / 4))]
    vo = [int((x / 2)) + (width / 2), int(hight + (y / 4))]
    ue = [int((x / 2)), int(int(hight * 1.25) + (y / 4))]
    uo = [int((x / 2)) + (width / 2), int((hight * 1.25) + (y / 4))]
    return ypos, ve, vo, ue, uo

def macroblock_per(yuv_frame, low_variance_threshold=1.25):
    offset = int(len(yuv_frame)/3)
    rows = len(yuv_frame) - offset
    columns = len(yuv_frame[0])
    y = yuv_frame[0:rows, 0:columns]
    u = yuv_frame[rows:rows+(offset/2), 0:columns]
    v = yuv_frame[rows+(offset/2):rows+offset, 0:columns]
```

FIGURE A6. ROIMBTruncation.py – This Script was written by William Oswald. This combines the OpenCV ROI Detection with the Macroblock Variance, and produces an output YUV video with both factors

Appendix A7: ROIMBTruncation.Py (cont. 2/3)

```
    face_cascade = cv2.CascadeClassifier('haarcascade_frontalface_alt2.xml')
    ROI_Index = FindFaces(cv2.cvtColor(yuv_frame, cv2.COLOR_YUV2BGR_I420), face_cascade)

    def macbroblock_per(sub_frame):
        MBtruncatedFrame = np.copy(sub_frame)
        bitsTruncated = 0
        for row_position in range(0, len(sub_frame), 16):
            for column_position in range(0, len(sub_frame[0]), 16):
                macroblock = []

                for j in range(row_position, row_position+16):
                    for i in range(column_position, column_position+16):
```

```
                        try:
                            macroblock.append(0.0001560911143834408 *
pow(int(sub_frame[j][i]), 2.628389343175764))
                        except IndexError:
                            break
                try:
                    avg_lum = sum(macroblock) / len(macroblock)
                    variance = sum([pow(byte - avg_lum, 2) / len(macroblock) for byte in
macroblock])
                except ZeroDivisionError:
                    pass
                else:
```

```
                    if variance >= low_variance_threshold:

                        for i in range(16):
                            for j in range(16):
                                #print(len(sub_frame))
                                if ((row_position + i) < len(sub_frame)):
                                    MBtruncatedFrame[row_position + i, column_position + j] =
TruncateIntValue(MBtruncatedFrame[row_position + i, column_position + j])
                                    bitsTruncated += 1


        return MBtruncatedFrame, bitsTruncated

    def repairROI(MBTruncatedImage, originalImage):
        ROIBitsSaved = 0
        MBROIFrame = np.copy(MBTruncatedImage)
        for (x, y, w, h) in ROI_Index:
            x_MB_Translated = int(x - (x % 16))
            y_MB_Translated = int(y - (y % 16))
            MB_width = math.ceil(w / 16)
            MB_hight = math.ceil(h / 16)

            for i in range(int(MB_width * 16)):
                for j in range(int(MB_hight * 16)):
                    pixel_x = x_MB_Translated + i
                    pixel_y = y_MB_Translated + j

                    for [image_x, image_y] in TranslatePositon(pixel_x, pixel_y, rows,
columns):
                        if (MBROIFrame[image_y, image_x] != originalImage[image_y, image_x]):
                            MBROIFrame[image_y, image_x] = originalImage[image_y, image_x]
                            #MBROIFrame[image_y, image_x] = 255
                            ROIBitsSaved += 1
```

Appendix A7: ROIMBTruncation.Py (cont. 3/3)

```
        return (MBROIFrame, ROIBitsSaved)
    uMBTruncatedFrame, bits2 = macbroblock_per(u)
    vMBTruncatedFrame, bits3 = macbroblock_per(v)
    yMBTruncatedFrame, bits1 = macbroblock_per(y)
    MBtruncatedframe = np.vstack((yMBTruncatedFrame, uMBTruncatedFrame, vMBTruncatedFrame))
    totalbits = bits1 + bits2 + bits3
```

```

    ROIMBFrame, totalsaved = repairROI(MBtruncatedframe, yuv_frame)
    return (ROIMBFrame, MBtruncatedframe, totalbits, totalsaved)

```

Appendix A8: WeightedPSNR.Py (cont. 1/5)

```
#!/usr/bin/python #Script written by Liam Oswald
import struct
import sys
import math
import pickle
import cv2
import numpy as np
from tqdm import tqdm
import os
import Macroblock
import ROIMBTruncation
from skimage.measure import compare_ssim

def psnr_rw(img1_ROI, img1_NROI, img2_ROI, img2_NROI, img3_ROI, img3_NROI, alpha):
    img1_ROI = img1_ROI.astype(np.float128)
    img1_NROI = img1_NROI.astype(np.float128)
    img2_ROI = img2_ROI.astype(np.float128)
    img2_NROI = img2_NROI.astype(np.float128)
    img3_ROI = img3_ROI.astype(np.float128)
    img3_NROI= img3_NROI.astype(np.float128)
    mse_img2_ROI = np.mean((img1_ROI - img2_ROI) ** 2)
    mse_img3_ROI = np.mean((img1_ROI - img3_ROI) ** 2)
    mse_img2_NROI = np.mean((img1_NROI - img2_NROI) ** 2)
    mse_img3_NROI = np.mean((img1_NROI - img3_NROI) ** 2)

    def Critical_Alpha(MSE_1_ROI, MSE_1_NROI, MSE_2_ROI, MSE_2_NROI):
        return (MSE_1_NROI - MSE_2_NROI) / (MSE_1_NROI + MSE_2_ROI - MSE_1_ROI -MSE_2_NROI)

    Crit_alpha = Critical_Alpha(mse_img2_ROI, mse_img2_NROI, mse_img3_ROI, mse_img3_NROI)
    D_frame_img2 = alpha * mse_img2_ROI + (1 - alpha) * mse_img2_NROI
    D_frame_img3 = alpha * mse_img3_ROI + (1 - alpha) * mse_img3_NROI
    psnr_rw_img2 = 20 * math.log10((255 / D_frame_img2))
    psnr_rw_img3 = 20 * math.log10((255 / D_frame_img3))
    return psnr_rw_img2, psnr_rw_img3, Crit_alpha

def psnr(img1, img2):
    img1 = img1.astype(np.float128)
    img2 = img2.astype(np.float128)
    mse = np.mean((img1 - img2) ** 2)
    if mse == 0:
        return 100
    PIXEL_MAX = 255.0
    return 20 * math.log10(PIXEL_MAX / math.sqrt(mse))

def WeightedPSR(frameNew, frameOld):
    hight = len(frameOld)
    width = len(frameOld[0])
    offset = int(len(frameOld)/3)
    rows = len(frameOld) - offset
    columns = len(frameOld[0])
    face_cascade = cv2.CascadeClassifier('haarcascade_frontalface_alt2.xml')
    ROI_Index = ROIMBTruncation.FindFaces(cv2.cvtColor(frameOld, cv2.COLOR_YUV2BGR_I420),
face_cascade)
```

FIGURE A8. WeightedPSNR.py – This Script was written by William Oswald. This calculates WPSNR for a YUV 422 video, and records the ouput into a file.

Appendix A8: WeightedPSNR.Py (cont. 2/5)

```
    columns = len(frameOld[0])
    face_cascade = cv2.CascadeClassifier('haarcascade_frontalface_alt2.xml')
    ROI_Index = ROIMBTruncation.FindFaces(cv2.cvtColor(frameOld, cv2.COLOR_YUV2BGR_I420),
face_cascade)

    def TranslatePositon(x, y, hight, width):
        # YUV file format explained here: https://answers.opencv.org/question/100149/how-to-
get-y-u-v-from-image/
        ypos = [x, y]
        ve = [int((x / 2)), int(hight + (y / 4))]
        vo = [int((x / 2)) + (width / 2), int(hight + (y / 4))]
        ue = [int((x / 2)), int(int(hight * 1.25) + (y / 4))]
        uo = [int((x / 2)) + (width / 2), int((hight * 1.25) + (y / 4))]
        return ypos, ve, vo, ue, uo

    def InROIRange(Pixelx, Pixely, faces):
        for (x, y, w, h) in faces:
            #translate ROI to macroblock edges
            x_u = int(x - (x % 16))
            y_u = int(y - (y % 16))
            w_u = int(math.ceil(w / 16) * 16)
            h_u = int(math.ceil(h / 16) * 16)
            if ((x_u < Pixelx < (x_u + w_u)) and (y_u < Pixely < (y_u + h_u))):
                return True
            else:
                return False

    def ExtractROIPixels(frameNew, frameOld, ROI_Index):
        inside_ROI_new = []
        inside_ROI_old = []
        outside_ROI_new = []
        outside_ROI_old = []
        picked_table = np.zeros((hight, width))
        for i in range(columns):
            for j in range(rows):
                ROI_flag = InROIRange(i,j,ROI_Index)
                locations = TranslatePositon(i, j, rows, columns)

                for [image_x, image_y] in locations:
                    if picked_table[image_y, image_x] == 1:
                        pass
                    elif (image_y >= hight or image_x >= (width)):
                        pass
                    elif (ROI_flag):
                        inside_ROI_new.append(frameNew[image_y,image_x])
                        inside_ROI_old.append(frameOld[image_y,image_x])
                    else:
                        outside_ROI_new.append(int(frameNew[image_y, image_x]))
                        outside_ROI_old.append(int(frameOld[image_y, image_x]))

                    picked_table[image_y, image_x] = 1

        inside_ROI_new = np.array(inside_ROI_new)
        inside_ROI_old = np.array(inside_ROI_old)
        outside_ROI_new = np.array(outside_ROI_new)
        outside_ROI_old = np.array(outside_ROI_old)
        return (inside_ROI_new, inside_ROI_old, outside_ROI_new, outside_ROI_old)
    inside_ROI_new, inside_ROI_old, outside_ROI_new, outside_ROI_old =
ExtractROIPixels(frameNew, frameOld, ROI_Index)
    psnr_ROI = psnr(inside_ROI_new, inside_ROI_old)
    psnr_not_ROI = psnr(outside_ROI_new, outside_ROI_old)
    return psnr_ROI,psnr_not_ROI
```

Appendix A8: WeightedPSNR.Py (cont. 3/5)

```
        for i in range(columns):
            for j in range(rows):
                ROI_flag = InROIRange(i,j,ROI_Index)
                locations = TranslatePositon(i, j, rows, columns)

                for [image_x, image_y] in locations:

                    if picked_table[image_y, image_x] == 1:
                        pass

                    elif (image_y >= hight or image_x >= (width)):
                        pass

                    elif (ROI_flag):
                        inside_ROI_new.append(frameNew[image_y,image_x])
                        inside_ROI_old.append(frameOld[image_y,image_x])
                        inside_ROI_M2.append(frameM2[image_y, image_x])
                    else:
                        outside_ROI_new.append(int(frameNew[image_y, image_x]))
                        outside_ROI_old.append(int(frameOld[image_y, image_x]))
                        outside_ROI_M2.append(int(frameM2[image_y, image_x]))
                    picked_table[image_y, image_x] = 1

        inside_ROI_new = np.array(inside_ROI_new)
        inside_ROI_old = np.array(inside_ROI_old)
        inside_ROI_M2 = np.array(inside_ROI_M2)
        outside_ROI_new = np.array(outside_ROI_new)
        outside_ROI_old = np.array(outside_ROI_old)
        outside_ROI_M2 = np.array(outside_ROI_M2)

        return (inside_ROI_new, inside_ROI_old, outside_ROI_new, outside_ROI_old,
inside_ROI_M2, outside_ROI_M2)
    inside_ROI_new, inside_ROI_old, outside_ROI_new, outside_ROI_old, inside_ROI_M2,
outside_ROI_M2 = ExtractROIPixels(frameNew, frameOld, frameM2, ROI_Index)
    psnr_rw_img2, psnr_rw_img3, Crit_alpha = psnr_rw(inside_ROI_old, outside_ROI_old,
inside_ROI_new, outside_ROI_new, inside_ROI_M2, outside_ROI_M2, alpha)

    return psnr_rw_img2, psnr_rw_img3, Crit_alpha

def Main(VideoOld, VideoNew, Old_method_Frame, xRez, yRez, path, alpha =0.9):
    # Get arguments
    filename = VideoOld
    xres = int(xRez)
    yres = int(yRez)

    frames_path = path + 'frames/'

    # Do OS operations
    #path = os.path.join(os.getcwd(), str(filename[:-4].rsplit('/', 1)[-1]))
    #path = os.path.join(os.getcwd(),'Results', str(filename[:-4].rsplit('/', 1)[-1]))
    #frames_path = os.path.join(os.getcwd(), str(filename[:-4].rsplit('/', 1)[-1])+"/frames/")
    try:
        #os.makedirs(path)
        os.makedirs(frames_path[:-1])
    except OSError:
        print ("Creation of the directory %s failed" % frames_path)
    else:
        print ("Successfully created the directory %s " % frames_path)

    # Read in the file
    capNew = Macroblock.VideoCaptureYUV(VideoNew, (yres, xres))
    capOld = Macroblock.VideoCaptureYUV(VideoOld, (yres, xres))
    capM2 = Macroblock.VideoCaptureYUV(Old_method_Frame, (yres, xres))
```

Appendix A8: WeightedPSNR.Py (cont. 4/5)

```
    filesize, framecount = capNew.file_statistics()
    print "Size of file in bytes: %d\nNumber of frames: %d\n" % (filesize, framecount)
    # Calculate average psnr and ssim for the whole video
    psnr_rw_img2_list = []
    psnr_rw_img3_list = []
    Crit_alpha_list = []
    for index in tqdm(range(framecount), unit="WeightedPSNR - ROI and non-ROI PSNR, critical
Alpha calc"):
        try:
            retOld, frameOld = capOld.read_raw()
            retNew, frameNew = capNew.read_raw()
            retM2, frameM2 = capM2.read_raw()

        except:
            print("ERROR")
            print(retOld)
            print(retNew)
            print(VideoNew)
            print("ERROR")
        frame_truncate_old = frameOld
        #CALCULATE ALL PSNR VALUES
        psnr_rw_img2, psnr_rw_img3, Crit_alpha = WeightedPSR_Two_Videos(frameNew, frameOld,
frameM2, alpha)

        psnr_rw_img2_list.append(psnr_rw_img2)
        psnr_rw_img3_list.append(psnr_rw_img3)
        Crit_alpha_list.append(Crit_alpha)

        #cap.display_raw_frame(frame, frames_path+str(filename[:-4].rsplit('/', 1)[-
1])+"_frame"+str(index)+".png")
        capNew.display_raw_frame(frame_truncate_old, frames_path+str(filename[:-4].rsplit('/',
1)[-1])+"_Original"+str(index)+".png")
    psnr_rw_img2_avg = np.nansum(psnr_rw_img2_list) /
np.count_nonzero(~np.isnan(psnr_rw_img2_list))
    psnr_rw_img3_avg = np.nansum(psnr_rw_img3_list) /
np.count_nonzero(~np.isnan(psnr_rw_img3_list))
    Crit_alpha_avg = np.nansum(Crit_alpha_list) /
np.count_nonzero(~np.isnan((Crit_alpha_list)))
    print len(Crit_alpha_list)
    # Write data to CSV file
    with open(os.path.join(path,str(filename[:-4].rsplit('/', 1)[-1])+".csv"), "wb") as file:
        file.write("Original Video,"+
                   "New Video,"+
                   ","+
                   "psnr_wr_OUR METHOD avg,"+
                   "psnr_wr_OLD METHOD avg,"+
                   "Critical_Alpha avg,"+

                   "," +
                   "," + "\n")
        file.write(str(VideoOld[:-4].rsplit('/', 1)[-1])+","+
                   str(VideoNew[:-4].rsplit('/', 1)[-1])+","+
                   str(',')+
                   str(psnr_rw_img2_avg) + "," +
                   str(psnr_rw_img3_avg) + "," +
                   str(Crit_alpha_avg) + "," +


                   #str(ssim_new_avg)+"\n\n\n")
                   str('') + "\n\n\n")

        file.write("Frame,psnr_wr_OUR METHOD,psnr_wr_OLD METHOD,Critical_Alpha, \n")
        for index, psnr_rw_img2_list, psnr_rw_img3_list, Crit_alpha_list in
zip(range(framecount),
```

Appendix A8: WeightedPSNR.Py (cont. 5/5)

```
        for index, psnr_rw_img2_list, psnr_rw_img3_list, Crit_alpha_list in
zip(range(framecount),
                                                                            psnr_rw_img2_list,
                                                                            psnr_rw_img3_list,
                                                                            Crit_alpha_list,

                                                                            #psnr_new,
                                                                            #ssim_new):
                                                                            ):

#file.write(str(index)+","+str(mb_y)+","+str(mb_u)+","+str(mb_v)+","+str(psnr_o)+","+str(ssim_
o)+","+str(psnr_n)+","+str(ssim_n)+"\n")
            file.write(str(index) + "," + str(psnr_rw_img2_list) + "," +
str(psnr_rw_img3_list) + "," + str(Crit_alpha_list) + "," + "\n")






if __name__ == "__main__":

    Main()
```

BIOGRAPHICAL SKETCH

William Oswald

Graduate and Undergraduate Schools Attended:

University of South Alabama, Mobile, Alabama.

Degrees Awarded:

B. S. Computer Engineering, University of South Alabama 2020

M. S. Electrical Engineering, University of South Alabama, 2021

Awards and Honors:

Graduate Research Assistant 2020-2021

Alabama EPSCOR Graduate Research Scholars Program Fellowship 2021-2022

IEEE ICECS 2021 Final Submissoin Reviwer