FPGA BASED HARDWARE ACCELERATOR FOR REAL TIME IMAGE COMPRESSION

By
Yicheng Zhou

Senior Thesis in Electrical Engineering
University of Illinois at Urbana-Champaign
Advisor: Thomas Moon

December 2024

Abstract

As imaging sensors evolve, so does the need for high-throughput data compression systems to handle increasing resolutions and frame rates. This thesis investigates the implementation of a hardware-accelerated encoder, designed in HDL for FPGA and ASIC platforms, providing a flexible, high-performance solution for various matrix-based data types. Additionally, a complementary decoder is developed in a high-level programming language to validate and demonstrate the encoder's functionality.

Subject Keywords: FPGA (Field-Programmable Gate Array), Image Processing, Real-Time Data Processing,

Contents

1 Introduction

Matrix compression is essential in various applications to reduce the data required for transmitting or storing information efficiently. A common example is video compression, widely used in streaming and storage. Traditionally, video encoding and decoding are performed by high-level languages on CPU/GPU-based systems. However, these systems face limitations, including reduced flexibility, limited scalability, and less predictable performance under varying loads. In contrast, FPGA/ASIC-based systems provide lower latency, greater energy efficiency, and a more self-contained design that many applications demand for high-speed, real-time data handling.

This study explores the feasibility of a hardware-based solution for video compression, emphasizing flexibility and scalability. By leveraging FPGA development techniques, this project implements critical components of video compression, such as discrete cosine transform, entropy encoding, and context-based intra-frame prediction. The design also integrates scalable, modular memory architectures to support the complex storage requirements of large-scale encoding.

This study aims to contribute to the field by demonstrating how FPGAs can be utilized to address key challenges in hardware-based compression. By focusing on modularity and adaptability, this study aims to offer insights into more accessi- ble and flexible FPGA design approaches, potentially paving the way for advanced development in hardware-accelerated data processing solutions.

2 Literature Review

The implemented algorithm is inspired by the H.264 standard developed by the Telecommunication Standardization Sector of the International Telecommunication Union (ITU-T)(1). It incorporates key components such as Context-Adaptive Variable Length Coding (CAVLC) for entropy coding and uniform scalar quantization for the quantization process.

In this approach, a large matrix is divided into smaller 4x4 matrices, commonly referred to as macroblocks (MBs). Each macroblock then undergoes several distinct compression stages, as illustrated in Figure 1.

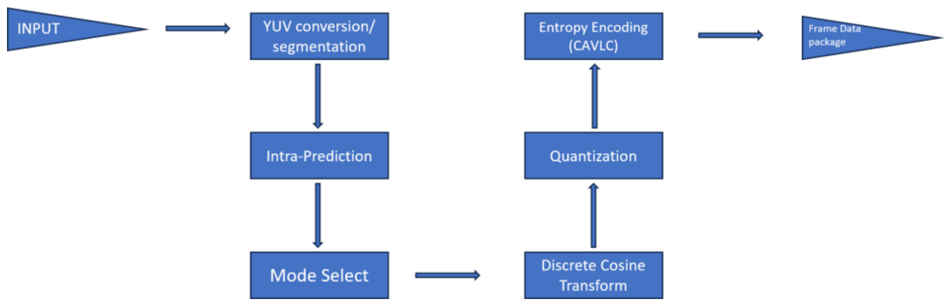

Figure 1: algorithm flow chart for each MB

2.1 Intra-prediction

This is the first stage of the compression process, which exploits spatial redundancy between macroblocks by predicting pixel values based on neighboring pixels. The predicted macroblock is then subtracted from the original resulting in a residual block that is encoded to reduce wasted space in the encoded data in the form of redundancy. Several prediction modes are utilized, and the mode that produces the smallest residual is selected for encoding.

- DC Prediction: Estimates the pixel based on the average of the value of available neighboring pixels, resulting in a uniform MB.
- Horizontal Prediction: Estimates the pixel based on the left neighbor pixel, very effective when the MB features horizontal textures.

- Vertical Prediction: Estimates the pixel based on the top neighbor pixel, very effective when the MB features vertical textures.
- Planar Prediction: Estimates the pixel value based on ta weighted average of neighboring pixels above and left, creating a smooth transitioning MB, effective for MB with gradual value changes.

2.2 Discrete Cosine Transform

The next stage after the intra-prediction is the Discrete cosine transform. This algorithm converts the residual block data from the spatial domain to the frequency domain. The idea is to concentrate the signal energy into a smaller number of coefficients, or in other words, generate a matrix that consists of mostly zeros which allow high compression ratio in the latter stages of the pipeline (entropy encoding)

$$X(u,v) = \frac{1}{4}\alpha(u)\alpha(v)\sum_{x=0}^{3}\sum_{y=0}^{3} x(x,y)\cos\left[\frac{(2x+1)u\pi}{8}\right]\cos\left[\frac{(2y+1)v\pi}{8}\right] \tag{1}$$

where

$$\alpha(u) = \begin{cases} \frac{1}{\sqrt{2}} & \text{if } u = 0, \\ 1 & \text{otherwise.} \end{cases} \tag{2}$$

Where X(u,v) is the DCT coefficient at frequency indices(u,v), x(x,y) is the pixel value at position (x,y) in the residual block, a(u) and a(v) are scaling factor which depends on the indices.

However, detailed cosine calculation is typically not feasible in systems that are time-critical or source-limited due to the nature of floating point operation. Thus it is typical to utilize an integer approximation. A Transform matrix is derived to approximate the floating point operation which is given by $C_{4\times4}$ here(2).

$$C_{4\times4} = \begin{bmatrix} 1 & 1 & 1 & 1 \\ 2 & 1 & -1 & -2 \\ 1 & -1 & -1 & -1 \\ 1 & -2 & 2 & -1 \end{bmatrix} \tag{3}$$

2.3 Quantization

The DCT coefficients would need to be quantized before entering entropy encoding. This step is designed to significantly reduce the amount of data needed to be processed by discarding less important frequency information. Though this would result in reduced precision, it is done in a way that is not humanly perceptible. This procedure is outlined in Equation 3.

$$Q(u,v) = \left\lfloor \frac{X(u,v)}{Q_{\text{matrix}}(u,v)} + 0.5 \right\rfloor \tag{4}$$

where Q_{matrix} is the quantization matrix specified by the standard (1). The division is element-wise, and rounding is done by adding 0.5 before applying the floor function.

2.4 Entropy Encoding

The entropy encoding method chosen here is the Context-Adaptive Variable Length Coding (CAVLC) is chosen from the H.264 standard [insert reference here]. The CAVLC is used to compressed the quantized DCT coefficients in a way that adapts to the data characteristics, since this is the most complicated algorithm used in the encoder, I will go into more detail.

- The algorithm starts by scanning the 4x4 matrix in a zigzag pattern shown in Figure 2, this allows the algorithm to rearrange the matrix into a 1D list, this ordering groups the significant (non-zero) coefficient at the beginning while placing the trailing zeros at the end.

- The CAVLC then counts the number of non-zero coefficients and the number of trailing ones in the list. Knowing the position and number of trailing ones enables CAVLC to use shorter codes for common cases, especially in smooth or flat regions with few non-zero coefficients.

- The non-zero coefficients are encoded using a combination of the number and the value of the non-zero coefficient, this format skips the zero values, reducing the data size significantly especially for matrices that have many zeros.

- The algorithm dynamically adjusts its variable-length coding tables based on the local context, such as the number of non-zero coefficients in neighboring blocks, this context adaptation allows the algorithm to assign shorter codes to frequently occurring values, improving compression efficiency.

- Finally, the algorithm assigns shorter variable-length codes to smaller, more frequent coefficient levels while using longer codes for larger values.

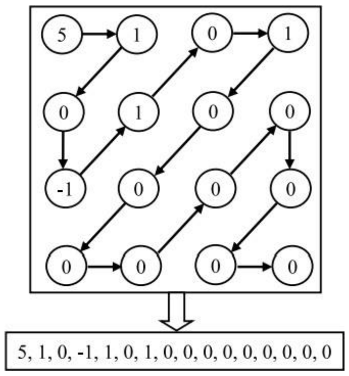

Figure 2: Zigzag ordering for flattening matrix (3)

3 Technical Overview

The encoder consists of four main components: the data input interface, data output interface, memory manager, and computation unit. For this project, a PC serves as both the data input and output interface, combining them into a single module. However, in practical applications, these interfaces can be separated to accommodate specific sensor control requirements. The FPGA-to-PC interface is implemented using the OpelKelly USB development kit. Data is initially stored in BRAM, accessible by both the interface and memory manager. Once data is passed to the computation unit, all temporary data and metadata are stored in distributed memory for faster access.

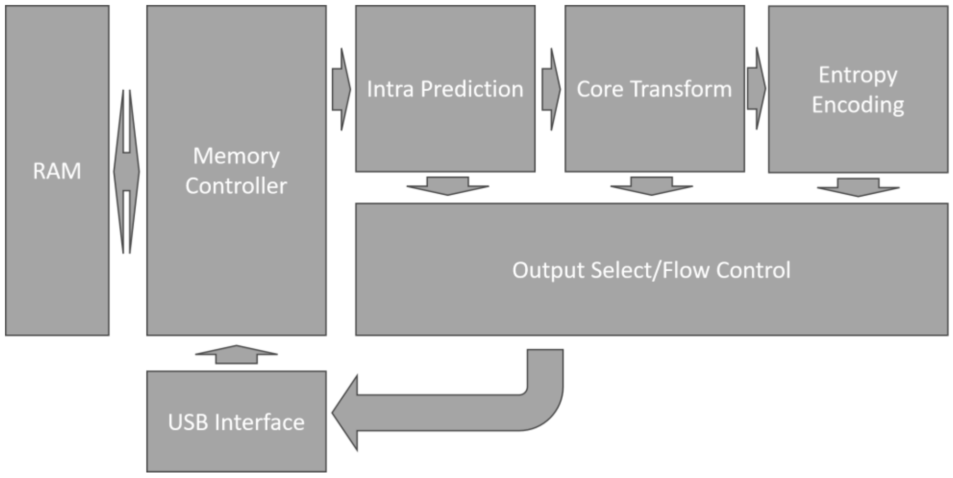

Figure 3: System Architecture

3.1 Interface

The USB interface is developed using the Opelkelly USB 3.0 development kit. To ensure safe data clock domain crossing, incoming and outgoing data are routed through FIFOs implemented with BRAM, chosen for its efficient use of LUT and FF resources. A block-throttled pipeline manages the FIFO, with a ready signal triggering the transmission of a specified data volume. Control signals and status reports are transmitted via asynchronous registers, where clock domain crossings are manually handled using double-flopping. While this technique is typically unsuitable for multi-bit signal synchronization, it is adequate here because the control and status signals are designed to be single-bit and level-sensitive, making cross-bit synchronization unnecessary.

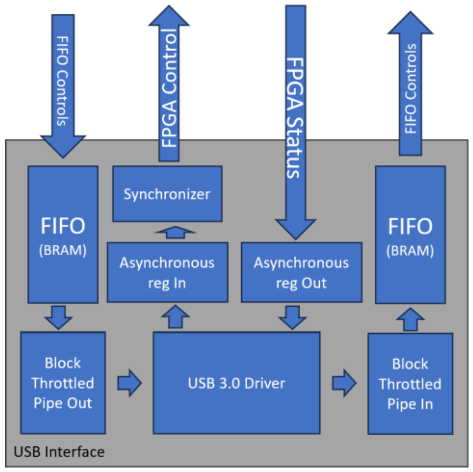

Figure 4: USB interface architecture

3.2 Memory Manger

The memory manager module is responsible for mapping the input data stream to specific memory addresses in a dual-port BRAM, organizing the data in a flattened index format for pixel-level access. This module receives data in 16-bit integer format and manages both read and write operations across BRAM's two ports. For input, it processes and assigns each incoming data stream to a unique memory address based on its flattened pixel index, ensuring that each pixel is accurately positioned in memory. Similarly, it organizes the output data stream by macroblock (MB) bits, converting the 2D macroblock layout into a 1D array format for storage in BRAM. The memory manager handles both ports, coordinating with the computation unit to process data when prompted, and interfacing with input and output ports to ensure smooth data flow. This setup allows efficient, high-speed access to memory for both data input and output while facilitating data retrieval for computation.

3.3 Computational Unit

The computational unit is designed to process each macroblock (MB) through a series of compression steps, including prediction, transformation, quantization, and

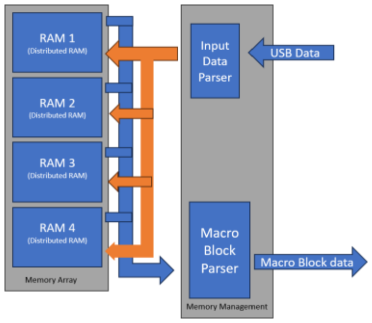

Figure 5: memory management

entropy encoding. The unit's main components are the predictor, Discrete Cosine Transform (DCT), quantizer, and entropy encoder. The predictor module is responsible for reducing spatial redundancy by estimating pixel values based on neighboring macroblocks; it includes four prediction modes as individual sub-components—each tailored to specific spatial patterns (such as DC, horizontal, vertical, and planar prediction). A top-level decision-making component selects the optimal prediction mode based on the resulting residuals, aiming to minimize data for encoding. Following prediction, the DCT module converts the residuals from spatial to frequency domain, where the quantizer then reduces data precision to emphasize visually significant information. The entropy encoder compresses the quantized coefficients into a compact bitstream. Throughout the process, the computational unit leverages distributed memory to store temporary data, ensuring quick access and efficient data handling across components.

3.3.1 Intra Prediction

The Intra Prediction module includes three modes: DC, Horizontal, and Vertical. Each mode is implemented independently. The DC mode requires four clock cycles due to critical path delay constraints. To maintain uniformity in data flow and synchronization, the Horizontal and Vertical modes are also designed to process in four cycles, even though they do not have the same delay limitations. This consistent timing ensures seamless integration with downstream modules, simplifying flow control.

Additionally, a finite state machine (FSM) is incorporated to manage data requests and parsing from the memory controller. The FSM also handles storing necessary context data into a cache for future use, optimizing access efficiency. The cache is implemented using registers to enable rapid data retrieval. Furthermore, a residual select module operates based on control signals from the asynchronous controller. It either performs direct routing via manual selection or computes the least total sum over four clock cycles, ensuring efficient processing.

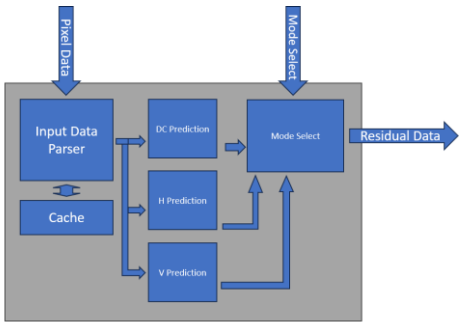

Figure 6: Intra Prediction Module

3.3.2 Core Transform

In the design, since there is no break points, DCT and quantize is grouped together and managed under coretransform. The DCT module performs a 4x4 Discrete Cosine trnaform operation, however due to hardware limitation, the computation did not strictly follow the mathematical formula. Instead, to avoid doing complicated floating point operation in hardware, an integer approximation is utilized as described by (2)

By applying integer arithmetic—through additions, subtractions, and shifts—it simplifies the implementation, allowing the use of efficient hardware components such as adders and shift registers. This integer-based approach ensures the DCT can be computed quickly and with fewer resources, making it suitable for embedded and real-time systems.

In practice, the 4x4 Integer DCT is often implemented using the butterfly algorithm, which breaks the 2D transform into two 1D stages—one applied to the rows and another to the columns. This reduces the complexity of direct matrix multiplication. After applying the butterfly algorithm, the resulting frequency coefficients can be quantized for compression. This method balances compression efficiency and computational cost, providing a practical and hardware-friendly approximation of the traditional 4x4 DCT.

Stage 1	**Stage 2**
$Y(0) = X(0) + X(3)$	$V(0) = Y(0) + Y(1)$
$Y(1) = X(1) + X(2)$	$V(2) = Y(0) - Y(1)$
$Y(2) = X(1) - X(2)$	$V(1) = Y(2) + (Y(3) \ll 1)$
$Y(3) = X(0) - X(3)$	$V(3) = Y(3) - (Y(2) \ll 1)$

Table 1: 4x4 forward 1D transform algorithm for H.264(2)

The quantization process in video compression follows the Discrete Cosine Transform (DCT) to reduce the precision of the DCT coefficients, enabling efficient data compression. After the DCT is applied, the resulting frequency coefficients are divided by a quantization matrix and then rounded to the nearest integer. The quantization matrix typically assigns lower precision to higher-frequency components, which often carry less perceptible visual information, thereby reducing their impact on image quality. This process significantly reduces the size of the data, but also introduces some loss of information, which is acceptable in video compression because the human eye is less sensitive to high-frequency details. The quantized coefficients are then further processed through entropy coding techniques, such as CABAC, to achieve high compression ratios.

3.4 Output Parser

The output parser module is responsible for managing various output modes and ensuring the correct formatting and transfer of data streams. It handles several modes, including raw data dumps for USB and memory verification, intra data dumps for intra prediction verification, core transform dumps, and the final encoded data stream. Each of these modes requires different data sizes, so one of the parser's main tasks is to dynamically reconfigure and parse the data stream into a FIFO structure, ensuring compatibility with the specific output requirements. Since the USB interface is block-throttled, the module also zeros out padding in the data stream when the final block does not meet the required size. Additionally, for entropy encoding, the module processes each macroblock (MB) individually, generating the corresponding metadata, which is transmitted alongside the encoded data. This ensures that each MB is properly encoded and can be reconstructed during decoding.

4 Experiment and Results

The experiment is conducted using the Xilinx Artix-7 series FPGA, with a PC serving to simulate data input and decode the encoded data stream via Python, chosen for its ease of implementation. As the primary focus of this study is on hardware, the performance of the PC component is not a critical factor. A random selection of images is utilized to evaluate the design's effectiveness.

Due to the limited resources ofthe Artix-7 family of FPGA, and the stress that a high resource usage rate would put on timing closure, the test unit is instantiated that only a single lane of the pipeline is instantiated. High LUT and LUT RAM are reported due to the reliance on distributed RAM, which is selected to reduce stall. The pipeline itself utilizes roughly 4000LUTs to instantiate. The system Resource usage is reported in the table below.

Resource	Utilization	Available
LUT	18382	47200
LUTRAM	12494	19000
FF	4317	94400
BRAM	44	105
IO	51	285
BUFG	4	32
MMCM	1	6

Table 2: Resource Utilization and Availability

The exact processing speed for the system is somewhat dependent on the availability of the USB port, USB transmission speed, and the resolution of the image. The test is conducted on a 256 × 256 resolution with opelkelly USB 3.0 driver, Thus both loading image and whole image process time is limited by the USB protocol at around 210us per image. A transition for an entire row of pixel data is depicted in the image below.

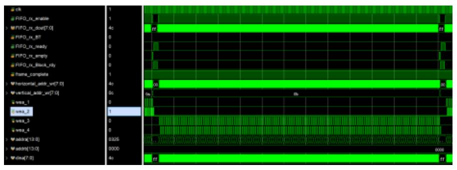

Figure 7: Timing Diagram for transmitting data

To validate the proposed algorithm, we leveraged an open-source Python library available online as a reference. While our implementation differs significantly from the H.264 standard in several aspects, we were able to borrow critical decoding components, such as inverse quantization and inverse CAVLC(4), to ensure compatibility and efficiency. For testing, we a ”standard” set of stock images as input data, which were transmitted to the FPGA for encoding. The encoded data was then received and processed on the PC for decoding and analysis. During testing, it was observed that certain compromises made to optimize the hardware implementation introduced some corrupted data points. However, these issues were not critical and could be effectively resolved through straightforward corrections in the software. This highlights the potential for seamless integration between hardware efficiency and software adaptability, ensuring robust overall performance. An example is listed In the Figure below.

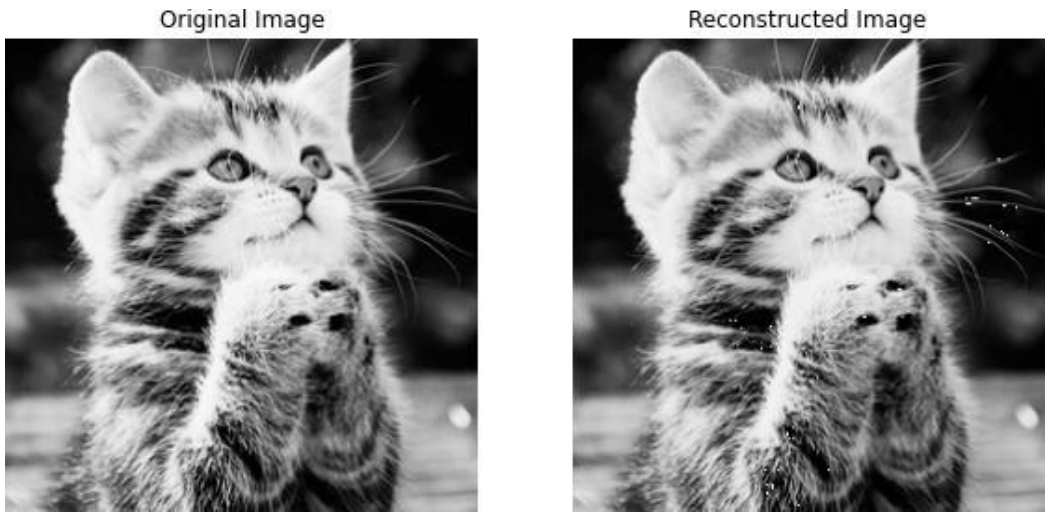

Figure 8: Raw and decoded image of stock image ”kitten”

5 Conclusion

In this thesis, we presented the design and implementation of a hardware-accelerated data compression system tailored for video encoding, leveraging the capabilities of the Xilinx Artix-7 FPGA. Through a systematic approach, we developed a computational unit that efficiently processes each macroblock by employing advanced techniques such as intra-prediction, Discrete Cosine Transform (DCT), quantization, and Context-Adaptive Variable Length Coding (CAVLC). The experimental results demonstrated the effectiveness of our design in achieving significant data compression while maintaining high-speed performance. This work not only contributes to the field of hardware-based video encoding but also highlights the potential of FPGA implementations in real-time applications.

6 Future Work

Looking ahead, several avenues for future work can enhance the capabilities of the proposed system. One significant improvement is the introduction of inter-frame prediction, which would allow the encoder to utilize temporal redundancy between consecutive frames, leading to even greater compression efficiency. Additionally, exploring alternative entropy encoding methods could provide flexibility and adaptability for different data characteristics, potentially improving performance for various types of video content. Furthermore, implementing parallel processing capabilities to handle multiple macroblocks simultaneously, rather than sequentially scanning through them, would significantly increase throughput and reduce latency, enabling real-time video processing applications. These enhancements would not only expand the functionality of the compression system but also position it as a competitive solution in the evolving landscape of video encoding technologies.

References

[1] I. T. Union, “Recommendation itu-t h.264: Advanced video coding for generic audiovisual services,” International Telecommunication Union, Tech. Rep., 2003, version 1. [Online]. Available: https://www.itu.int/rec/T-REC-H.264

[2] A. T. Bunji Antoinette Ringnyu, “Implementation of forward 8x8 integer dct for h.264/avc frext,” *The Eurasia Proceedings of Science, Technology, Engineering Mathematics (EPSTEM)*, vol. 1, pp. 353–358, 2017.

[3] G.-L. J. G.-L. J. e. a. Fuentes-Alventosa, A., “Cavlcu: an efficient gpu-based implementation of cavlc,” *J Supercomput 78*, vol. 7556–7590 (2022), 2022. [Online]. Available: https://link.springer.com/article/10.1007/s11227-021-04183-8#citeas

[4] Wattery, “Pycodec,” 2019. [Online]. Available: https://github.com/Watterry/PyCodec/tree/master