VHDL MPEG-1 Video Compression Algorithm

Lucky Douangmanivong, Ryan White, Trey Nguyen, Jaehyun Kim, Jamie Castro, Eric Ly,
Mohamed El-Hadedy (ECE 4304/L Professor)
luckyd@cpp.edu, rswhite@cpp.edu, tmnguyen@cpp.edu, jaehyunkim@cpp.edu,
jaimecastro@cpp.edu, egly@cpp.edu, mealy@cpp.edu
Department of Electrical and Computer Engineering
Cal Poly Pomona
ece@cpp.edu

***Abstract* – One of the most widely used lossy video compression algorithms in products and technology today is known as MPEG-1. Over the years from its creation in 1992, it has been developed and improved, but is a constant standard, in theory, for all video compression algorithms alike. MPEG-1 algorithms are efficient and reliable when implemented within the computer architecture of any Field Programmable Gate Array (FPGA) board. As computer engineers, it is our job to understand how algorithms such as MPEG-1 works, as well as being able to properly demonstrate its features. This experimentation will explore many topics within video compression algorithms, such as the mathematical design of them as well as some of their platform limitations.**

***Keywords* – Computer Architecture, DCT, FPGA, Image Compression, lossy video compression, MPEG-1**

I. INTRODUCTION

MPEG-1 is a video and audio compression standard that was created in 1992 to compress VHS video and audio. This would allow for VHS media to be put on more platforms such as cable TV and video CDs. The MPEG-1 format would compress the video and audio to around 1.5 Mbits/s which allows for these products to be used in a large amount of products. MPEG-1 is still a popular format today and is widely known for popularizing the MP3 audio format. This compression format was also extended to MPEG-2, MPEG-3, etc.

The creation and implementation of MPEG-1 Image Compression Algorithms may seem complex, but can be broken down into a set of smaller processes that need to be completed. The very first that needs to be done is to convert the RGB values, which is what the CPU sees to display frames of a video, into YCbCr values. This color space of Luma, Chroma Blue, and Chroma Red is used widely in video systems. For consumers, this is the color space that is commonly referred to. Many Video Compression Algorithms are based on the H.261 standard of using reference frames to forward predict, but MPEG-1 uses future frames as backward prediction.

The next step in MPEG-1 Video Compression Algorithms is to apply a Discrete Cosine Transform (DCT) to an 8x8 block of uncompressed pixel values. First, a Forward Discrete Cosine Transform converts the uncompressed block into an indexed array of frequency coefficient values. These coefficient values are then quantized by dividing them by a larger step size and rounding to an integer value. Again, a quantization matrix is created as a result, which tells the encoder how important each piece of visual information is. Each value in the matrix then corresponds to a certain frequency component of the compressed video image.

After the 8x8 matrix is quantized by the DCT algorithm, a zig-zag scan is performed on the matrix to generate a bitstream of data that will be encoded with run length encoding (RLE). This method of encoding creates a method with which the system skips the zeros found within the bitstream and signals to the system where the next significant value is located which reduces the size of the data.

The paper is organized as follows:
Sect. II - Related Work
Sect. III - Mathematical Model of Image Comp.

Sect. IV - Platform Limitations
Sect. V - Experiments
Sect. VI - Conclusion

II. RELATED WORK

When MPEG-1 was developed in the early 1990s, people were not aware of how many improvements would be made to them. However, as viewers began expecting higher quality of video, many companies jumped at the idea of making MPEG-1 better. There were many different improvements made, but essentially, the next few video compression algorithms that advanced the area of industry were the MPEG-2 model, MPEG-4 model, H.261 model, and H.263 model.

MPEG-2 features a much higher picture quality than MPEG-1 and could supply up to 5 full bandwidth channels. This is because MPEG-2 focused on TV transmission and other applications capable of 4 Mbps and higher data rates. MPEG-2 was the first sign of a breakthrough in video compression algorithms, but still proved insufficient at the time to provide better video resolution and more digital channels in TV.

Around the end of the 1990s, MPEG-4 rose in popularity, as many were looking for a way to code at very low bit rates for wireless telecommunications. It is a superset of MPEG-1 and MPEG-2, creating a common communication language to describe tools and algorithms for coding of objects rather than of separate algorithms.

Another video compression algorithm, H.261, was developed specifically for video conferencing. It is based on using a discrete cosine transform (DCT) and allows for fully encoding INTRA-frames and INTER-frames. H.261 does not do as much compression as MPEG model algorithms, but is intended for telephony and can minimize encoding and decoding delay while achieving a fixed data rate.

We know that from the history of video compression algorithms, each one was created for different purposes. However, they each brought their own improvements and concepts to video compression, allowing us to further the quality and efficiency of our data today. Even in the future, we will continue to use these ideas to improve more aspects of video compression, and these new types of video compression techniques can be seen in the following graph.

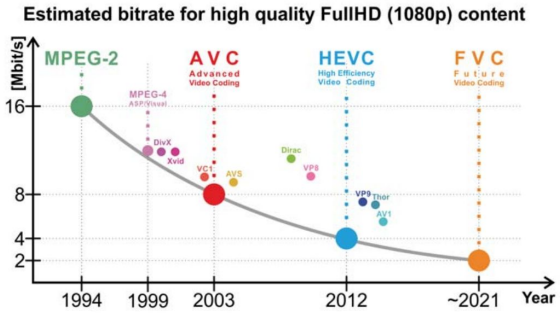

Figure 1 Quality of Video Compression Algorithms since 1994

III. MATHEMATICAL MODEL

There are a lot of mathematical calculations that are done in order to create the MPEG-1 Compression. First, there are the dimensions for the frames that are used to make the video. These frames can go up to 4096X4096 pixels but are normally set at 360X240 pixels. This is similar to using a JPEG image for every frame in the video. The MPEG-1 also discards certain areas of the picture in order to compress the video and to have the human eye fully perceive the video. Before the video gets encoded, the colo-space is turned into Y'CBCR with Y being Luma, Cb being Chroma Blue, and Cr being Chroma Red. The chroma is subsampled to 4:2:0 which means that half of the resolution vertically and horizontally are reduced. There are also equations that are used to quantize the image and these equations are in the form of matrices. This is so that the picture can be quantized like a grid with coordinates. The equation for quantization for MPEG-1 format is $Xq[i,j] = Round(X[i,j]/Q[i,j])$ while the inverse quantization equation is $X'[i,j]=Xq[i,j]*Q[i,j]$.

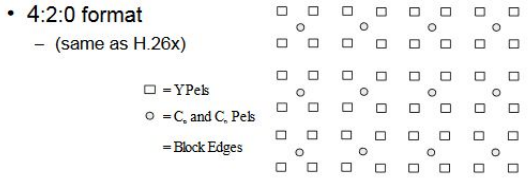

Figure 2 Y'CBCR Formatting with 4:2:0 Subsampling

IV. PLATFORM LIMITATIONS

The limitations of our hardware were not from the board itself, but rather from time constraint

when implementing the Verilog code onto the board. In order to fully implement our module onto the board, we needed to either create a ROM module that was initialized with all of the frame data to be compressed, or to create a module that allowed for UART communication between the PC and the FPGA to stream the data into the compressor. Both of these solutions to our problem required knowledge and development that we did not have enough time to work with. We instead chose to create a testbench and simulate the results of the hardware deployment on the FPGA and acquired the results from there.

Another limitation we had was the amount of lookup tables and flip-flops on the Nexys DDR 4 FPGA. If we were to fully deploy the entire MPEG-1 compression standard with capabilities of compressing I, P, and B frames; we would exceed the resources available on the board. We were able to deduce this from the metrics acquired by constructing the I-frame verilog code and we acquired about 38,300 LUT's and 34,600 FF. Since that was for simply one frame type, we can assume that the implementation of all 3 frame types would increase these numbers by at least double. Our solution to this was to utilize a PYNQ Z1 FPGA and attempt to run python code that was accelerated by snippets of HDL that would increase performance. We did not have enough time to implement this method, but it is a possibility to be noted for future work.

V. EXPERIMENTS

Experiments were conducted by first ensuring that we had a baseline to compare our hardware implementation against. This involved the creation/acquisition of Python code that would compress a series of 24 frames to compress as I-Frames. We then ran the Python code on a PC and used its run time of 3 minutes as the baseline.

The software pulled 24 frames from a one second clip of the open source Blender Foundation movie "Big Buck Bunny" and downloaded them as PNGs. The images were then run through a script that converted them from rgb to 4:2:0 chroma-subsampling YCbCr bitstreams cropped to a smaller frame size. A third script would then be run to read this bitstream in and compress the data using discrete cosine transform, JPEG quantization tables, and huffman coding for final compression. The compressed blocks and their huffman tables are then serialized out to another bitstream.

The next step in our experiment was to implement the same code onto the Pynq board and see how long it would take to run the code on the FPGA. After deploying the code on the chip, we acquired a run time of about 38 minutes. The discrepancy of run time with the PC is most likely due to the fact that the python code acts the same on both systems, with the PC having better hardware specs which accounts for the faster execution time.

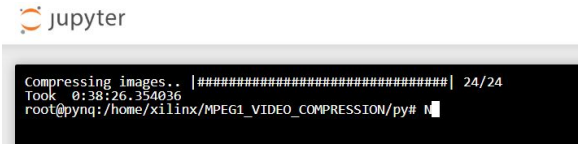

Figure 3 Execution Time on Pynq Z1

Finally our last metric to be measured is the runtime for the actual hardware implementation of the I-Frame encoding. Our group had difficulty deploying our module on the chip so, in order to provide deliverables by the deadline, we opted to utilize a testbench simulation to acquire a theoretical execution time. We did this by converting the same 24 frames as the python code into raw hex values which we then saved as a text file. The test bench was then coded to read this text file and stream the hex data into an 8x8 matrix with which we send onto our conversion module to be converted into their respective Luma and Chroma values. From there the full intra frame compression is executed onto the data and we acquire the encoded bitstream as well as the execution time from the test bench output. After testing we acquired a 33 minute runtime to run the simulation, with 12 minutes of actual cpu time. The Simulation end time was about 0.024 seconds which means that the theoretical time to encode all of the frames is significantly faster than the python code. This is most likely due to hardware being completely dedicated to the processing of the frames and there is no overhead related to executing processes that are non-essential to the compression of the frames that a pc would have to execute.

```
c36b673c
f35c5d7d
96d2dff0
ff0038cd
08cab9f5
cf5fd7e9
fe71556c
d42b9691
55864202
7692a460
71d33962
0fb8cf23
35dc2dd0
73b88c73
b472ad8e
dc9e7d87
7ef5fbac
3a7f278e
4673d33c
9c8fc871
40000fc0
65bb7a8e
$finish called at time : 24377900 ns
```

Figure 4 Testbench Completion Time

VI. CONCLUSION

In conclusion, the motivation of this project was to demonstrate understanding of both software and hardware performance metrics by implementing the MPEG -1 standard of video compression. We did this by first implementing a python code to compress I-frames and ran it on both a PC and a PYNQ Z1 FPGA. When comparing both systems, the PC greatly outperformed the PYNQ board due to the PC's more robust processor. This performance imbalance quickly changed when we started phase 2 of our testing which was to attempt to implement a purely hardware version of the compressor by deploying verilog code on the board. We were unsuccessful at fully deploying the code; however, we were able to acquire theoretical metrics from utilizing a test bench. From the Verilog testbench, we acquired a theoretical execution time that was shorter than that of the PC run Python code. This increase in performance demonstrated the viability of using hardware to accelerate specific jobs within a system.

FUTURE WORK

For future work, we decided that we could improve the code for both hardware and software in order to make it run more smoothly and efficiently. We also could have enhanced the code to be used for other operations such as the extended MPEG-2 and MPEG-4 formats to compare the run times and emphasize the improvements from one to the other. Another area for future work would be implementing the Verilog code on the board which would be done by creating ROMS with frames for storage and creating UART serial communication to stream the RGB values. Another area of improvement for this project would have been to manage our efforts better and have more consistent communication between group members, as that is always an area that everyone can work on.

ACKNOWLEDGEMENTS

We would like to acknowledge xilinx for allowing us to use the Pynq-z1 FPGAs to develop and test this project on MPEG-1 Image Compression. We would also like to thank our professor, Dr. Mohamed El-Hadedy for constant support and guidance.

REFERENCES

[1] A. Davis, *An Overview of Video Compression Algorithms,* February 3, 1998, Accessed on: April 11, 2020. [Online]. Available: https://www.eetimes.com/an-overview-of-video-compression-algorithms/#

[2] F. Gedegast, *MPEG-FAQ: multimedia compression*, June 02, 1996, Accessed on: April 8, 2020. [Online]. Available: https://ieeeauthorcenter.ieee.org/wp-content/uploads/IEEE-Reference-Guide.pdf

[3] H. Purnhagen, *MPEG-1: coded storage of sampled sound waves*, November 07, 2001, Accessed on: April 8, 2020. [Online]. Available: https://sound.media.mit.edu/resources/mpeg4/audio/faq/mpeg1.html

[4] Karwowski, Damian & Grajek, Tomasz & Klimaszewski, Krzysztof & Stankiewicz, Olgierd & Stankowski, Jakub & Wegner, Krzysztof. (2017). 20 Years of Progress in Video Compression – from MPEG-1 to MPEG-H HEVC. General View on the Path of Video Coding Development. 525. 3-15. 10.1007/978-3-319-47274-4_1.

[5] *MPEG-1*, March 12, 2020, Accessed on: April 12, 2020. [Online]. Available: https://en.wikipedia.org/wiki/MPEG-1