Pipelined Hardware Video Compressor & Decompressor

SDMay24-12

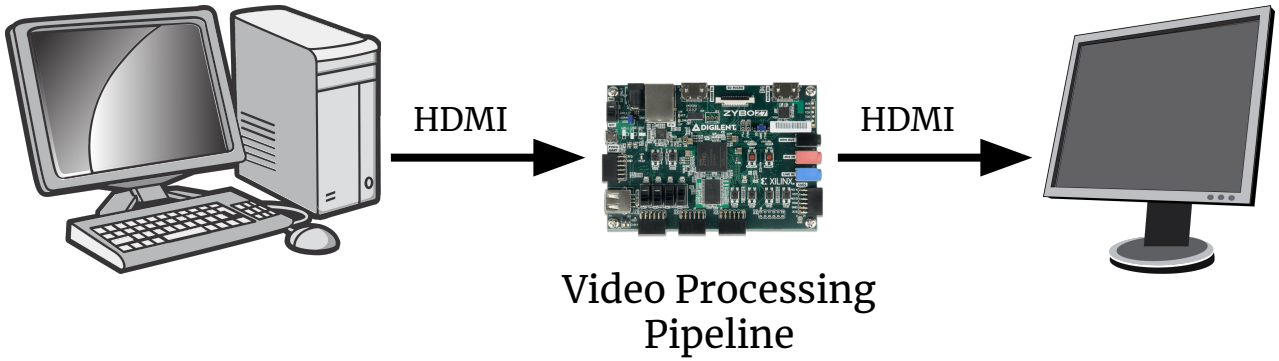
Logan McDermott, Kareem Eljaam, Caleb Rock, Benjamin Meinders, Colsen Selk

IOWA STATE UNIVERSITY

Introduction

Client: John Deere

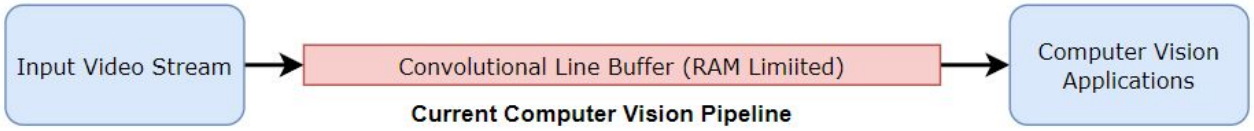
Project Goal: Create a fully pipelineable image compression and decompression scheme that is mappable to an FPGA. The goal is to take live HDMI video input into an FPGA, run compression and decompression, then output the new video stream to HDMI output to observe the difference

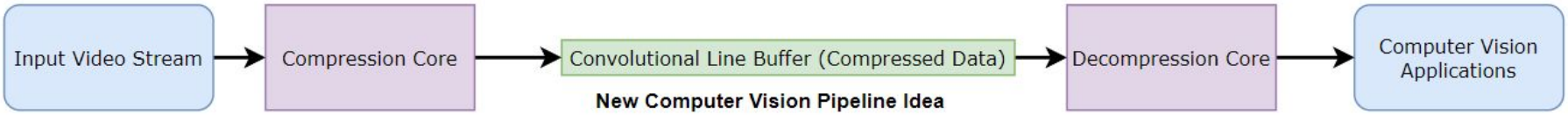


Context/Problem Statement

Problem: Computer vision applications rely on near-zero latency when processing fast convolutions which is achieved with a fully-pipelined FPGA design. The FPGA however, is limited by the amount of RAM it has available for storing convolutional line buffers

Solution: Implement a compression core for the input of each line buffer and a decompression core for the output of each line buffer that runs on a live video stream

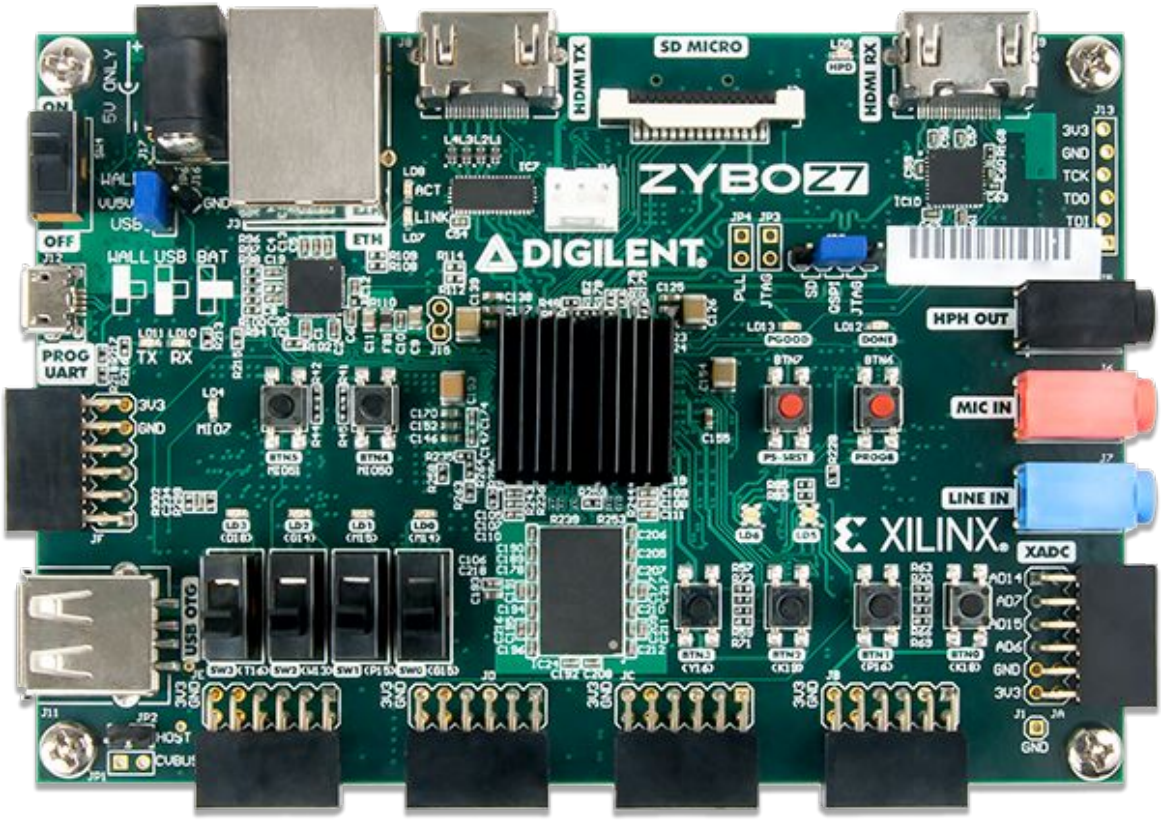




Requirements

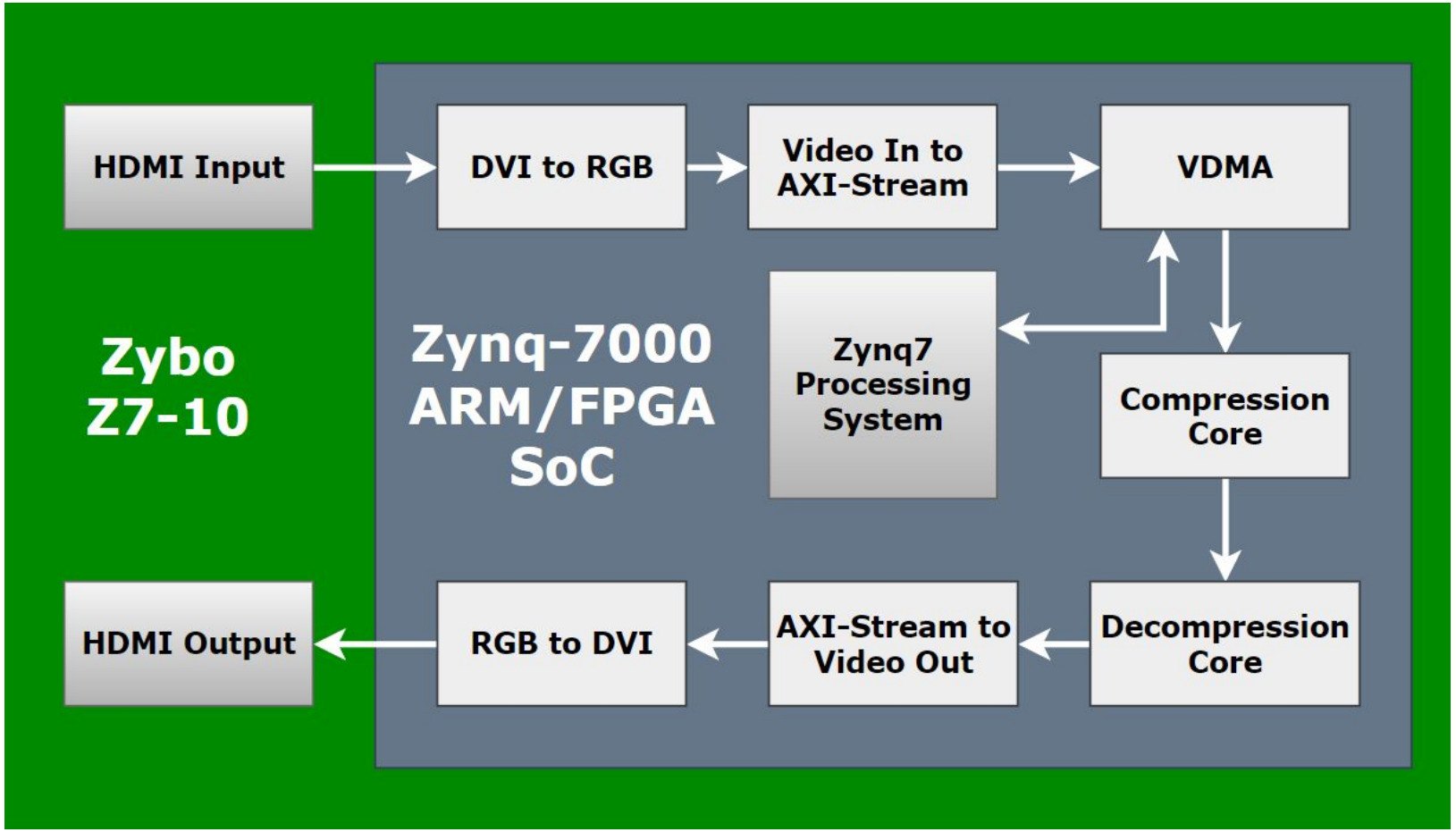
- Compression and decompression are to be performed on a live input stream.
- The latency should be low enough to allow for near-zero latency for computer vision processing.
- Prioritize logic simplicity and pipelineability over compression ratio.
- Each encoded pixel must be of a predetermined size, as a key requirement for a fully pipelineable design.

Hardware Decision



- **Zybo Z7-10**
- HDMI RX and HDMI TX
- Zynq-7000 ARM/FPGA SoC Development Board
- Tutorial for HDMI passthrough from Digilent to get started

Hardware Design



Failed Approaches

Purchasable IP Core

- Sold by AMD
- Very expensive
- Few compatible with Zynq-7000

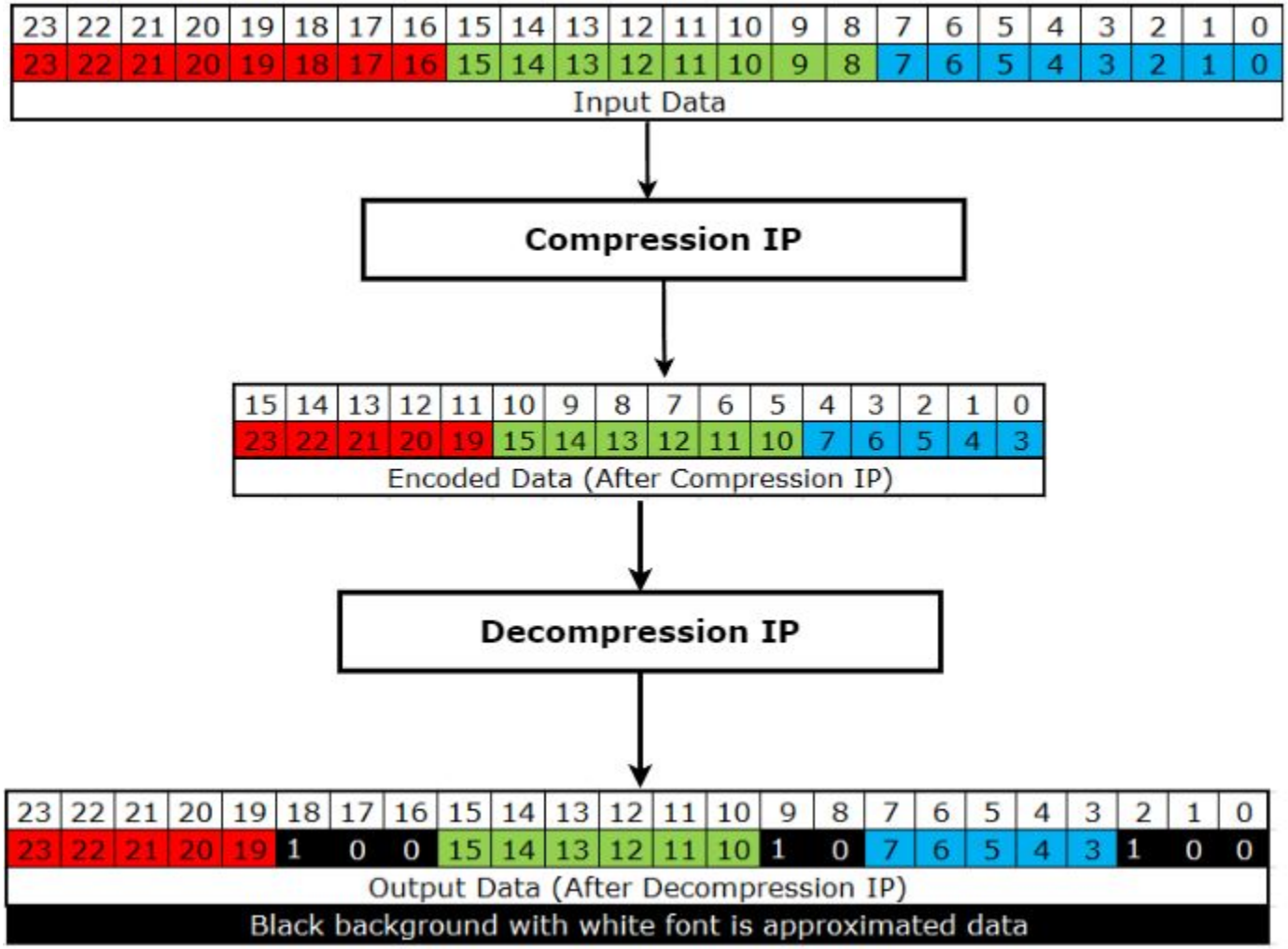
Open Source IP Core

- Opencores
  - Website for open source IP cores
  - Compatibility issues
- CCSDS123 Algorithm
  - Decompression unavailable
  - Intended use was for 3D images.

Vitis HLS

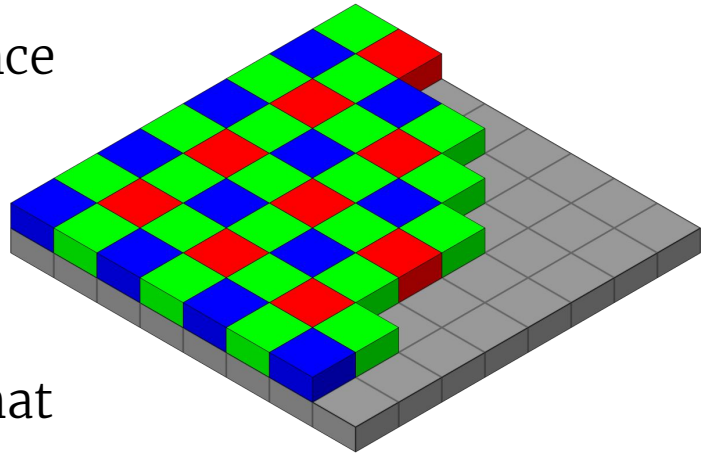
- High level synthesis
- Generated IP
- Not compatible with our HDMI passthrough design (required system clock to be the same as pixel clock).

Compression Scheme



Compression Details

- There are many ways to convert from RGB24 to RGB16
  - Examples: 4:6:6 & 5:5:6
  - All three colors cannot have same number of bits
- We chose 5:6:5 as our new color space
  - Human eyes are most sensitive to variance in green light
  - Similar to Bayer filter being 50% green



- By only eliminating insignificant bits, the Worst case scenario for recovering data is that the red and blue values are off by 4 and the green values are off by 2
- Number of possible colors before: 16,777, 216 —> 65,536 colors

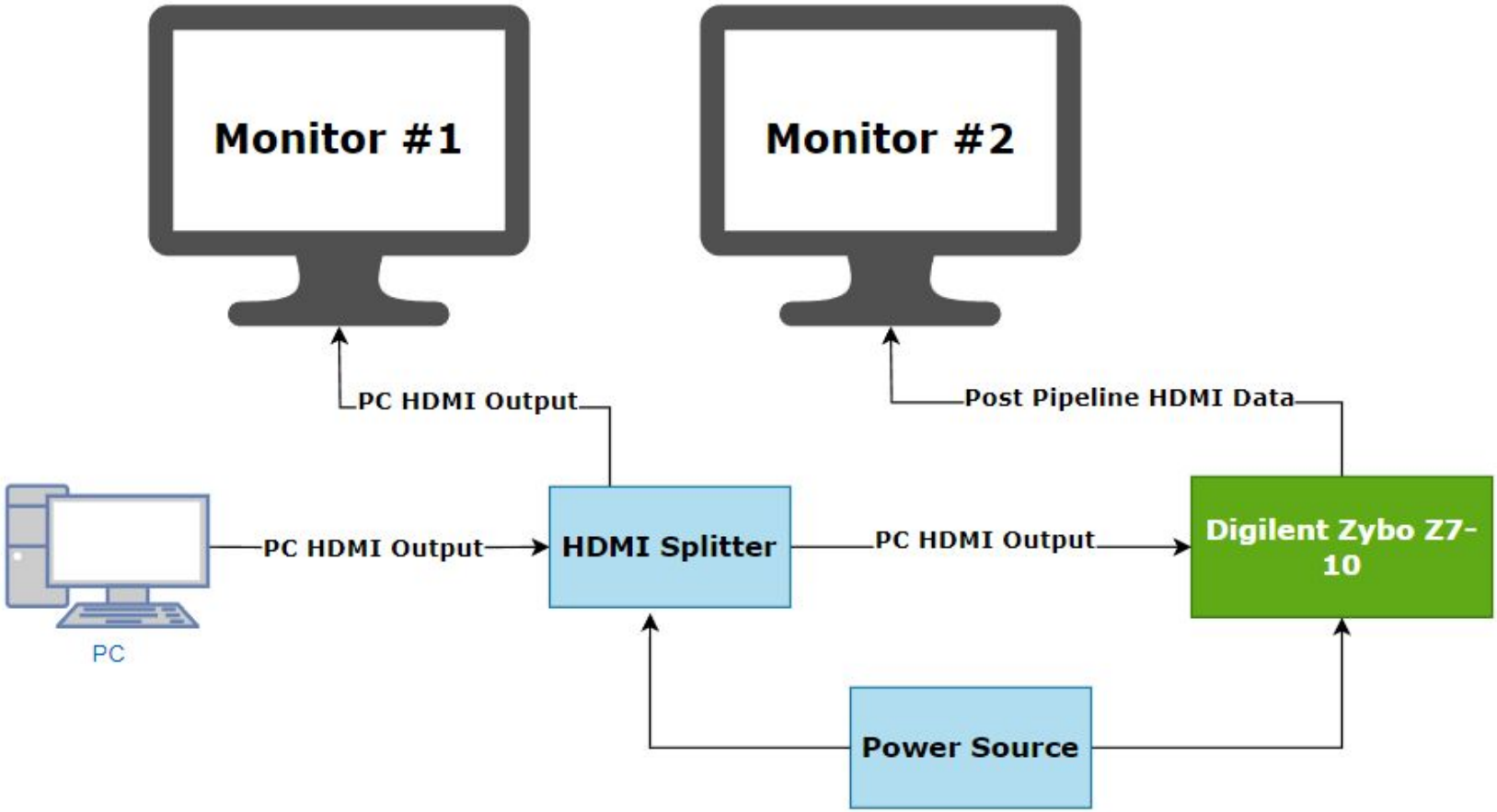
- To back up our choice for choosing 5:6:5, we wrote a script to test the rate of occurrence for the last three bits on a byte
- Found that the results were near uniform with a slight bias towards 000 (0) or 111 (7)
- Resulted in picking 4:2:4 respectively for the values to append on the 5:6:5 (median)

```
0: 0.124899563852
1: 0.124341329373
2: 0.124785325763
3: 0.123800735059
4: 0.12432885906
5: 0.124254303511
6: 0.124820590429
7: 0.128769292952

0: 0.128842048433
1: 0.124118122156
2: 0.126125923065
3: 0.127215007042
4: 0.126066811005
5: 0.124180389161
6: 0.121124126159
7: 0.122327572979

0: 0.134274798536
1: 0.131365429009
2: 0.1321702723
3: 0.129357901037
4: 0.123617870485
5: 0.11869804975
6: 0.114662062191
7: 0.115853616692
```

Demonstration Design



Demonstration

Testing/ Results

Software Testing / LZW

- LZW was our first failed approach
- Thrives off of repetitive data
- The algorithm was not used for implementation due to a slow bitstream
- Was incapable of handling ASCII encodings above 127 which was interpreted as negative bytes even with unsigned byte as data type
- Tried to implement BWT into the algorithm to increase repetitive values

Hardware Visual Testing / Results

Video Before Pipeline

Video After Pipeline

Compression Ratio = Size of Original Data = 1.5
Size of Compressed Data

Amount of RAM needed is reduced to 66% of the original requirement

Hardware Latency Testing / Results

Hardware Throughput Testing / Results

$$Throughput = Pixel_{Freq} \times \frac{\#\ Pixels}{cycle} \times \frac{\#\ Bytes}{Pixel} = 134\ MHz \times 1 \times 3 = 402\ \frac{MB}{s}$$

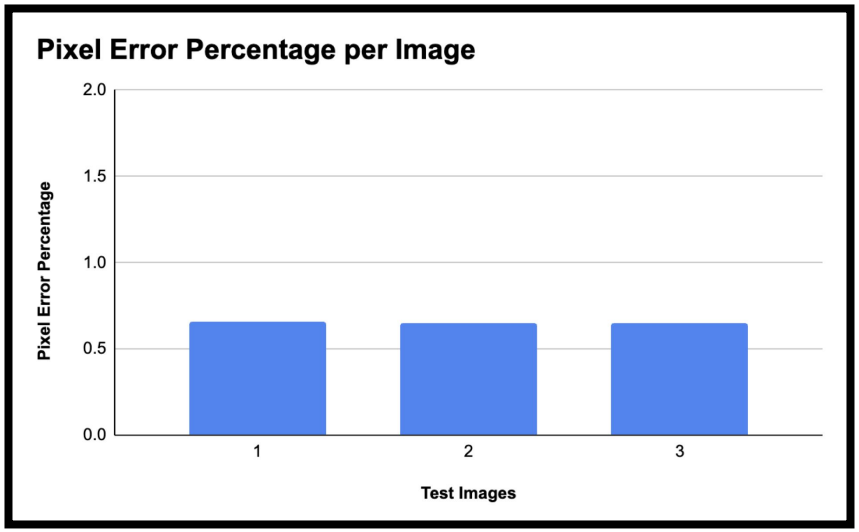
- Pixel Clock is driven by the Processing System at 134 MHz
- All three bytes of RGB 8:8:8 per pixel are compressed in parallel so we calculate 3 bytes per pixel
- Our overall video throughput supports 402 MB/s with our current design
- For context, 4k video only requires around 25 MB/s

Data Loss Metrics / Testing

$$Pixel_{Error\%} = (Pipeline_{Pixel} - Original_{Pixel}) \times \frac{1}{256} \times 100$$

```
for pxlw in range(width):
  for pxlh in range(height):
    # Extract rgb values from pixel map twice
    r, g, b = rgb[pxlh, pxlw]
    rt, gt, bt = rgb[pxlh, pxlw]
    # Alter the numbers like compression algorithm
    rt = ((rt >> 3) << 3) + 4
    gt = ((gt >> 2) << 2) + 2
    bt = ((bt >> 3) << 3) + 4
    # Add to variance to total
    totalVarR += (abs(r - rt)) / 256
    totalVarG += (abs(g - gt)) / 256
    totalVarB += (abs(b - bt)) / 256
# Divide variance of each variable by px count for average
totalVarR = totalVarR / (width * height)
totalVarG = totalVarG / (width * height)
totalVarB = totalVarB / (width * height)
```

Data Loss Metrics / Results




```
Image1.NEF
Total Pixel Count: 24304952
Total Variance Red: 0.7993278167654888
Total Variance Green: 0.39044459373443735
Total Variance Blue: 0.7769971693937103
Total Variance per Pixel: 0.6555898599645454

Image2.NEF
Total Pixel Count: 24304952
Total Variance Red: 0.7857966718315265
Total Variance Green: 0.3903381660916261
Total Variance Blue: 0.7857019926947397
Total Variance per Pixel: 0.6539456102059641

Image3.NEF
Total Pixel Count: 24321024
Total Variance Red: 0.7841839006758105
Total Variance Green: 0.3899334686730295
Total Variance Blue: 0.7801492451808362
Total Variance per Pixel: 0.6514222048432254
```

➔ Note: .NEF are RAW image files that usually take up > 25 MB

Conclusion

- Delivered a successful implementation of the requirements
- Provided a foundation for another team to work from
- Obtained information regarding best strategies for implementing compression algorithms
- Learned how to face roadblocks, change course, and still deliver value