

Giaan Nguyen

Lab 5: JPEG Decoder Design on FPGAs

ECEN 689-600: FPGA Information Processing Systems  
Thursday, March 11, 2021

We wish to make a zigzag scanner and a Huffman coder that will both be used towards a provided JPEG decoder. As the zigzag transformation is merely a reordering of an image's indices, we only perform simulations for the Huffman coder as seen in Figure 1. The output values in Figure 1 are similar to those provided by the TA, with the exception of zeroing the Huffman length for each reset. This should not matter, as we will see in the .bmp files generated later.

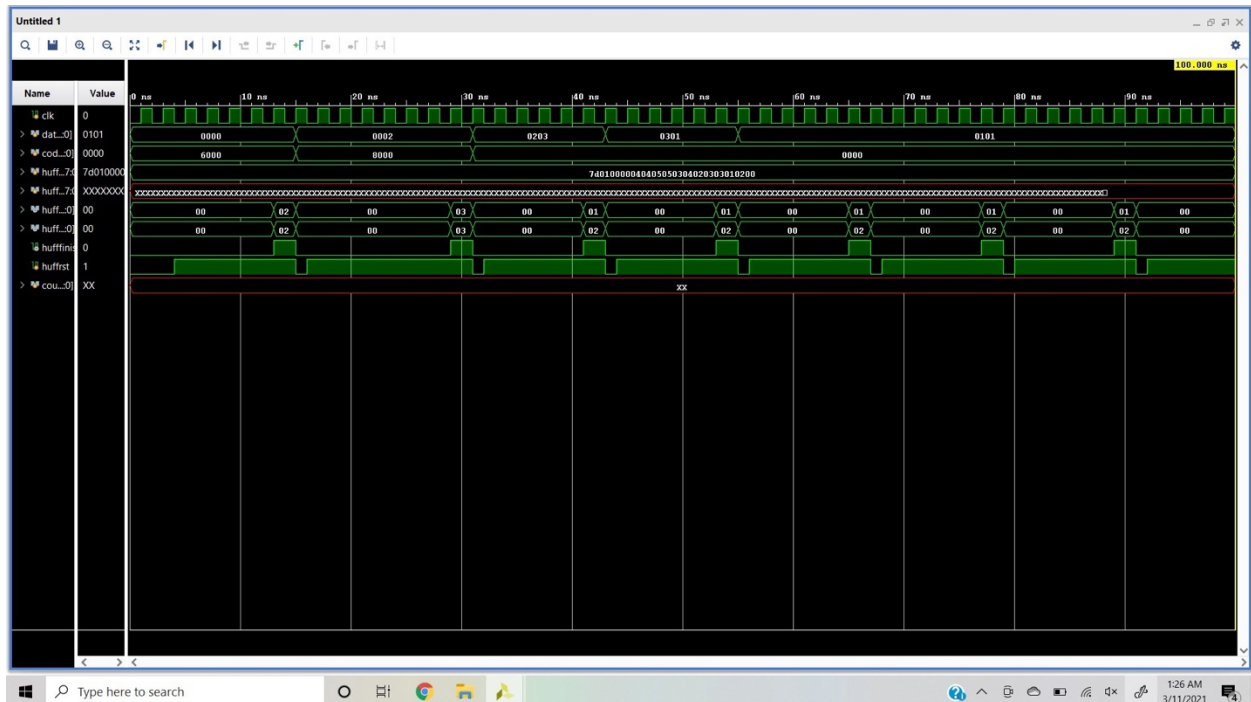


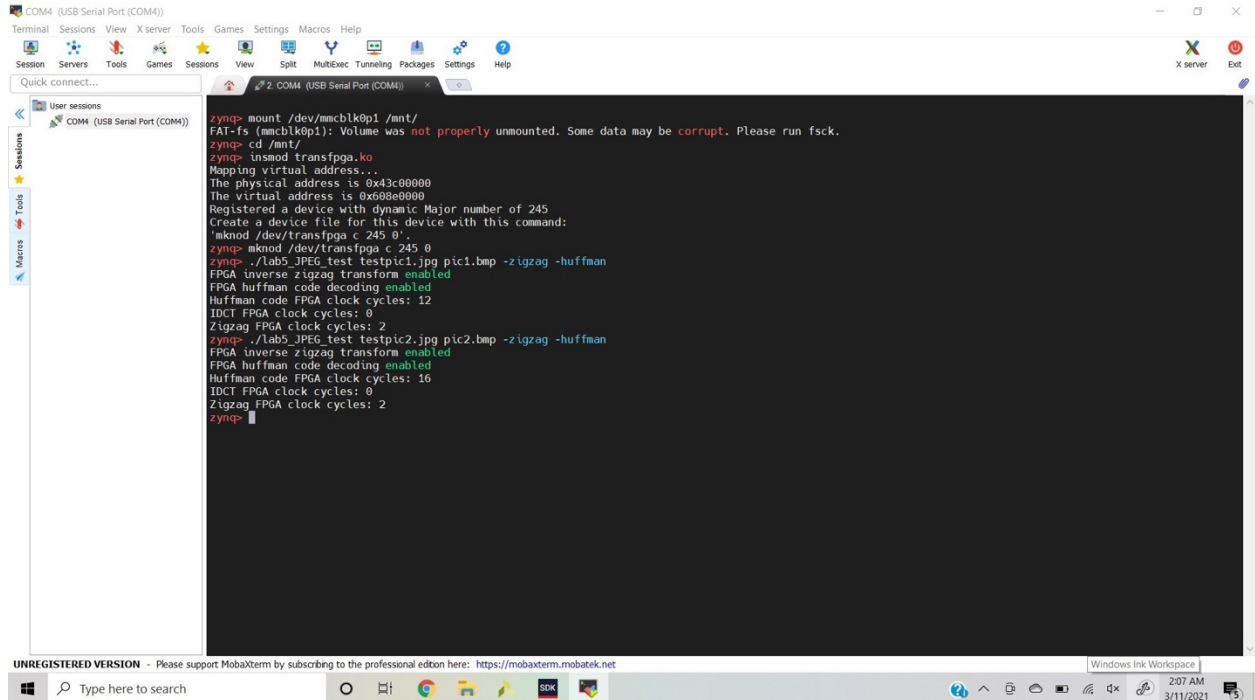
Figure 1. Simulation for the Huffman coder module.

## FPGA Outputs

The .bmp files generated by the JPEG decoder (with zigzag and Huffman coding modules added) can be seen in Figure 2 as well as in the submitted zipped folder. In Figure 3, the terminal outputs for the JPEG decoder is displayed. For “testpic1.jpg”, we can see that the number of clock cycles for the Huffman is 12, whereas “testpic2.jpg” underwent 16 clock cycles for Huffman. For both tests, zigzag used 2 clock cycles, and 0 clock cycles were recorded for the inverse discrete cosine transform (IDCT).



Figure 2. Generated BMP files from the JPEG decoder.



```
zynq> mount /dev/mmcblk0p1 /mnt/
FAT-fs (mmcblk0p1): Volume was not properly unmounted. Some data may be corrupt. Please run fsck.
zynq> cd /mnt/
zynq> insmod transpga.ko
Mapping virtual address...
The physical address is 0x43c00000
The virtual address is 0x608e0000
Registered a device with dynamic Major number of 245
Create a device file for this device with this command:
'mknod /dev/transpga c 245 0'
zynq> mknod /dev/transpga c 245 0
zynq> ./lab5_JPEG_test testpic1.jpg pic1.bmp -zigzag -huffman
FPGA inverse zigzag transform enabled
FPGA Huffman code decoding enabled
Huffman code FPGA clock cycles: 12
IDCT FPGA clock cycles: 0
Zigzag FPGA clock cycles: 2
zynq> ./lab5_JPEG_test testpic2.jpg pic2.bmp -zigzag -huffman
FPGA inverse zigzag transform enabled
FPGA Huffman code decoding enabled
Huffman code FPGA clock cycles: 16
IDCT FPGA clock cycles: 0
Zigzag FPGA clock cycles: 2
zynq>
```

Figure 3. FPGA output for the JPEG decoder, printed in a terminal.

## Implementation Summary

Figure 4 shows the power, timing, and utilization summaries for the implementation of the JPEG decoder with added zigzag and Huffman modules.

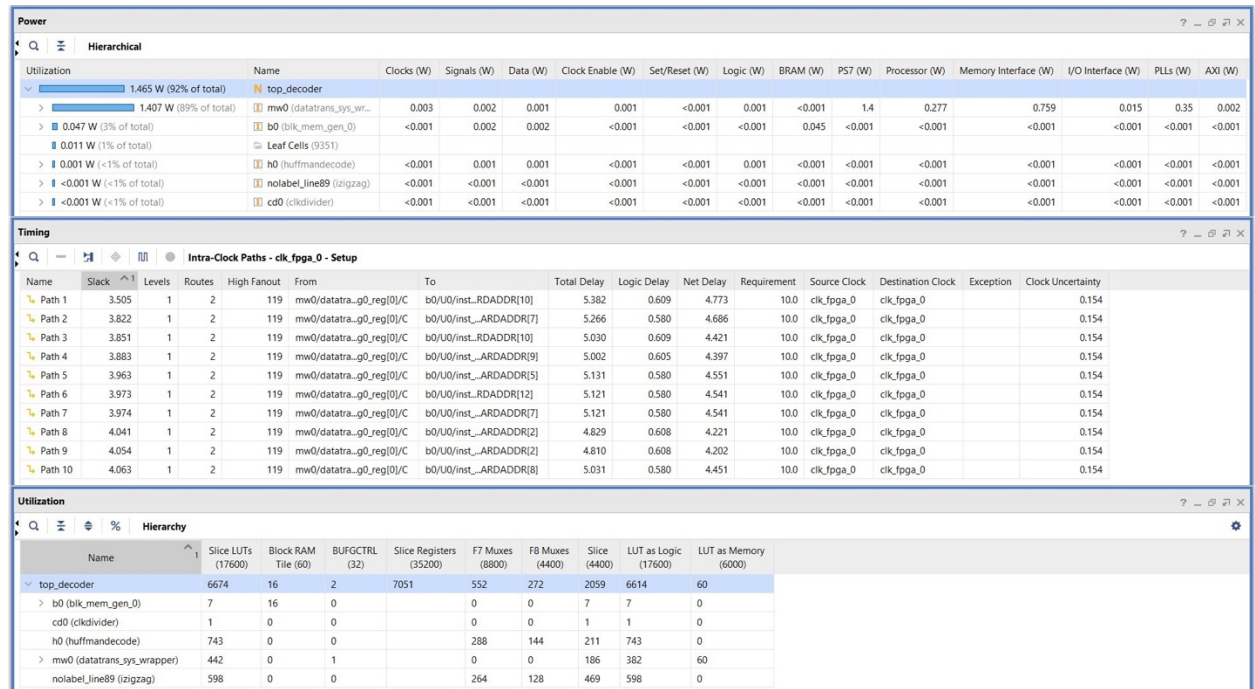


Figure 4. Implementation summary for the JPEG decoder: power (top), timing (middle), and utilization (bottom).

## Questions

### 1. What is the purpose for using zigzag transformation in JPEG encoding?

In a typical DCT matrix, the frequency content increases along each row and each column. For sake of efficiency, zigzag is done so that the lower frequency coefficients in the top left corner can be grouped together just as the higher frequency coefficients in the bottom right corner can be grouped together. This should intuitively make sense as pixels within a block (or neighborhood) are more likely to be similar to each other than pixels along a row or column. Additionally, a grouping of 0s due to the zigzag can be compressed; that is, the zigzag also exploits redundancy.

### 2. What is the purpose for using Huffman encoding in JPEG encoding?

Huffman coding is used in image compression to assign each image intensity value (can be thought of as a char) a code that would minimize the image size. That is, for chars with higher probabilities of appearing in an image, they can be assigned codes with shorter lengths; inversely, chars with lower probabilities are assigned codes with longer lengths. The idea is that the total number of bits due to the reassignments from Huffman coding would result in a smaller file size than the original.